Generic SNMP Device Management User Guide and Toolkit

Document 1316

Notice

Copyright Notice Copyright © 2002 - present by Aprisma Management Technologies, Inc. All rights reserved worldwide. Use, duplication, or disclosure by the United States government is subject to the restrictions set forth in DFARS 252.227-7013(c)(1)(ii) and FAR 52.227-19.

Liability Disclaimer Aprisma Management Technologies, Inc. (“Aprisma”) reserves the right to make changes in specifications and other information contained in this document without prior notice. In all cases, the reader should contact Aprisma to inquire if any changes have been made.

The hardware, firmware, or software described in this manual is subject to change without notice.

IN NO EVENT SHALL APRISMA, ITS EMPLOYEES, OFFICERS, DIRECTORS, AGENTS, OR AFFILIATES BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS MANUAL OR THE INFORMATION CONTAINED IN IT, EVEN IF APRISMA HAS BEEN ADVISED OF, HAS KNOWN, OR SHOULD HAVE KNOWN, THE POSSIBILITY OF SUCH DAMAGES.

Trademark, Service Mark, and Logo Information SPECTRUM, IMT, and the SPECTRUM IMT/VNM logo are registered trademarks of Aprisma Management Technologies, Inc., or its affiliates. APRISMA, APRISMA MANAGEMENT TECHNOLOGIES, the APRISMA MANAGEMENT TECHNOLOGIES logo, MANAGE WHAT MATTERS, DCM, VNM, SpectroGRAPH, SpectroSERVER, Inductive Modeling Technology, Device Communications Manager, SPECTRUM Security Manager, and Virtual Network Machine are unregistered trademarks of Aprisma Management Technologies, Inc., or its affiliates. For a complete list of Aprisma trademarks, service marks, and trade names, go to:

http://www.aprisma.com/manuals/trademark-list.htm.

All referenced trademarks, service marks, and trade names identified in this document, whether registered or unregistered, are the intellectual property of their respective owners. No rights are granted by Aprisma Management Technologies, Inc., to use such marks, whether by implication, estoppel, or otherwise. If you have comments or concerns about trademark or copyright references, please send an e-mail to spectrum-docs@aprisma.com; we will do our best to help.

Restricted Rights Notice (Applicable to licenses to the United States government only.)
This software and/or user documentation is/are provided with RESTRICTED AND LIMITED RIGHTS. Use, duplication, or disclosure by the government is subject to restrictions as set forth in FAR 52.227-14 (June 1987) Alternate III(g)(3) (June 1987), FAR 52.227-19 (June 1987), or DFARS 52.227-7013(c)(1)(ii) (June 1988), and/or in similar or successor clauses in the FAR or DFARS, or in the DOD or NASA FAR Supplement, as applicable. Contractor/manufacturer is Aprisma Management Technologies, Inc. In the event the government seeks to obtain the software pursuant to standard commercial practice, this software agreement, instead of the noted regulatory clauses, shall control the terms of the government's license.

Virus Disclaimer Aprisma makes no representations or warranties to the effect that the licensed software is virus-free. Aprisma has tested its software with current virus-checking technologies. However, because no antivirus system is 100-percent effective, we strongly recommend that you write protect the licensed software and verify (with an antivirus system with which you have confidence) that the licensed software, prior to installation, is virus-free.

Contact Information Aprisma Management Technologies, Inc., 273 Corporate Drive, Portsmouth, NH 03801 USA

Phone: 603.334.2100
U.S. toll-free: 877.468.1448
Web site: http://www.aprisma.com

Contents

Notice 2

Preface 11

Intended Audience 11

Text Conventions 11

Document Feedback 12

Online Documents 12

Overview 13

The GnSNMPDev Management Module 14

Overview 14

Modeling 15

Identification 15

Interfaces 16

Applications 16

Views 17

Traps, Events and Alarms 17

Customizing the GnSNMPDev Management Module 19

Adding Support for Additional MIBs 19

Adding Support for Additional Alerts, Events and Alarms 20

Adding SpectroWATCHes to Track and Manage Model Conditions 20

Adding Device Type Values for Device Identification 20

Mapping a System Object Identifier to a Device Type 21

The Device Identification List 22

Unregistered Device List 24

Device Type Identification in a Distributed Server Environment ..25

Working in a Fault Tolerant Environment 27

Customization Considerations 27

Developing a New Management Module 30
Representing the Device with Model Types 30
Additional MIB Support 31
Unique Device Model Icon and Symbol 31
Unique SpectroWATCH 32
Device Identification 32
Unique Trap Mapping 33
Interface/Port Model Creation 33

Appendix A: Creating a New Device Model Type 34
Designing a New Device Model Type 34
Creating a New Device Model Type 37
Model Type Flags 38
Configuring a New Device Model Type 38
Setting Attribute Values 38
Ensuring SPECTRUM Selects the Appropriate Device Model Type and Provides Identification 41
SysOIDVerifyList/DeviceNameList Discovery and Identification Mechanism 42
Vendor_Object_ID/Desc_Key_Word Discovery and Identification Mechanism 43
System_Desc_Verify/Desc_Key_Word Discovery and Identification Mechanism 44
Discovery and Identification Flowchart 44
Configuring Serial Number Handling 46
Saving Your Changes 46
Creating SpectroGRAPH Support Files for a New Device Model Type 46
Customizing a New Device Model Type 46
Distributing a New Device Model Type 47

Appendix B: Creating a New Application Model Type 48
Understanding Derivation Points and Model Fragments 48
Choosing a Derivation Point 51
Board and Port Considerations 52

Port–Oriented Devices 52
Chassis Devices 52
Creating Application Model Types 55
Import Required MIBs 56
Derive the Application Model Type 56
Setting the default_attr or the default_attr_list 57
Set the Model_Name 59
Map Model Fragments 59
Set Model Type Flags 60
Saving Your Changes 60
Modeling Ports and Boards 61
Modeling Boards with GnModule 61
GnModule Attributes 61
GnModule Icons 61
Deriving from GnModule 61
Modeling Ports with GnPort 62
GnPort Icons 62
Creating New Port Model Types 62
Port and Board Model Information 62
Creating SpectroGRAPH Support Files for a New Application Model Type 63
Distribution 63

Appendix C: Creating SpectroGRAPH Support Files for New Model Types 64
Running mmbuild 64
Deleting Support Files 66

Appendix D: Customizing Views and Device Models 67
Customizing Device Icon Symbols in the SpectroGRAPH 67
How Images are Selected for GnSNMPDev Device Icon Symbol ..68
The image_index Attribute 69
How the Value of the image_index Attribute is Determined ...69
Modifying Support Files 72

.Bas and .OPR Files 72
DYNIM Files 73
Using a Custom Image 75
Device Icons in Application Views 77
Disabling Automatic Device Image Selection 78
Customizing Alarm Manager and Search Manager Device Icon Symbols 80
Device Model Appearance 80
Associating the Model Type with the Icon 80
Defining the Device Icon Symbol Template 81
Customizing Menu Selections in the SpectroGRAPH 83
Customizing Menu Selections in the Alarm Manager and the Search Manager 83
Creating an .isv File 83
Mapping your Model Type to the .isv File 88
Implementing Conditional Menu Selections 88
Method 1: Optional Menu Selection 89
SpectroGRAPH Icons 89
Alarm Manager and Search Manager Icons 89
Method 2: Hidden Sub-Menu Selection 90
SpectroGRAPH Icons 90
Alarm Manager and Search Manager Icons 90
Method 3: Application Existence Menu Selection 91
Customizing Views 92
Controlling View Display with the CsViewControl File 92
Generic Information Block (GIB) Views 93
Device View 94
Adding Device Views 94
Accessing the Device Views 95
How the Device View Works 95
The Action Mechanism 96
Device Topology View 99
Accessing the DevTop View 99

How the DevTop View Works 100
Dealing with Double-Width Boards 103

Appendix E: Alert, Event and Alarm Processing 105
Alerts 105
Events 106
EventDisp File 106
Event Format Files 107
Alarms 107

Appendix F: SpectroWATCH 108

Appendix G: Distribution 109
Creating an Index File 109
Running mkmm 109
Running mkcd 110

Appendix H: Model Fragment Reference 111
The GnChassis_MF Model Fragment 111
aChassisManager (IM) 112
deviceMh_Attr (IM) 112
resetOnChange (2) 112
configInterval (1,2) 112
boardIndex_Attr (1,2) 113
boardTerm (1,2) 114
boardType_Attr (1,2) 114
boardType_Map (2) 115
boardName_Attr (1,2) and nameParsingCmds (1,2) 118
boardName_Map (1,2) 119
modulesType_Attr (IM) 120
modulePibPrefix (1,2) 121
slotCount_Attr (1, 2) 124
chassisType_Map (1,2) 126
blankPibIndex (1) 127
The GnDeviceIO_MF Model Fragment 128

devicesMh_Attr (IM) 129
GnDeviceIO_MF Attributes 130
GnDataRelay_MF Attributes 132
The GnDataRelay_MF Model Fragment 135
managersMh (IM) 136
useMapping (2) 136
portMap (2) 137
groupIndex_Attr (1,2) 139
groupTerm (1,2) 139
portIndex_Attr (1,2) 140
portType_Attr (2) 141
portType_Map (2) 142
altPibPrefix (1,2) 145
portName_Map (1,2) 145
portAttachRel (2) 147
The GnPortUI_MF Model Fragment 147
counter_Attr (1,2) 148
status_Attr (1,2) 148
statusEnum_VTC (1,2) 148

Appendix I: Relations 150
Overview 150
Lost and Found Intelligence 150
Implementing Lost and Found Intelligence for New Relations ... 152

Appendix J: Tutorials 153
Tutorial 1: Modeling Non-Data Relay MIBs 153
Purpose of this Tutorial 153
Creating the Application 153
Importing the Liebert UPS MIB 154
Creating the Application Model Type 154
Setting the default_attr_list Attribute 155
Tutorial 2: Modeling Port-Oriented Devices 157
Purpose of this Tutorial 157

Port-Oriented Device Application View Model 158
Port-Oriented Device View 159
Port-Oriented Device Topology View 160
Before You Begin 161
Gathering MIB Information 161
Building the Application Model Type 164
Creating an Application Model Type 164
Setting Up the Application Model Type 165
Naming the Port Model 165
Setting the default_attr Attribute 165
Adding the GnPortUI_MF Model Fragment 166
Defining Device Configuration Management 166
Modeling the Ports 167
Defining the Port Display 169
Testing the Port Application Model 171
If the Test Fails 172
Tutorial 3: Building a Hub Manager Application 173
Purpose of this Tutorial 173
Hub Manager Application View Model 174
Hub Manager Chassis Device View 175
Hub Manager Chassis Device Topology View 176
Gathering MIB Information 177
Chassis Information 177
Repeater Information 178
Port Model Information 179
Building the Hub Manager Application 180
Model Type 180
Creating a Hub Manager Application Model Type 180
Setting Up the Hub Manager Application Model Type 181
Naming the Hub Manager Application Model 182
Setting the default_attr Attribute 182
Defining the Chassis 183
Setting the Data-Relay Functionality 184

Testing the Hub Manager Application Model 185
If the Test Fails 186
Labeling the Boards 187
Using a Descriptor 187
Using a Map 190
Modeling the Repeater Ports 191
Defining the Repeater Element 191
Defining the Port Display 192

Index 195

Preface

In this section:

Intended Audience [Page 11]

Text Conventions [Page 11]

Document Feedback [Page 12]

Online Documents [Page 12]

Intended Audience

This guide is intended to explain the function of SPECTRUM's generic management module, GnSNMPDev, at three different levels:

- As a generic management module that can be used to represent an SNMP-compliant network device that does not have a corresponding SPECTRUM management module.
- As a customizable management module to which you can add new views, SpectroWATCHes, device type information, or additional MIB support.
- As a toolkit to create new management modules to support devices that are not SNMP-compliant or require complex customizations.

Text Conventions

The following text conventions are used in this document:

Element	Convention Used	Example
User-supplied parameter names	*Courier and Italic* in angle brackets <>.	The user needs to type the password in place of *<password>*.
On-screen text	Courier	The following line displays: path="/audit"
User-typed text	Courier	Type the following path name: C:\ABC\lib\db

Element	Convention Used	Example
Cross-references	Underlined and hypertext-blue	See Document Feedback [Page 12].
References to SPECTRUM documents (title and number)	*Italic*	*SPECTRUM Installation Guide (0675)*
Functionality enabled by SPECTRUM Alarm Notification Manager (SANM)	SANM in brackets [].	[SANM] AGE_FIELD_ID

Document Feedback

Please send feedback regarding SPECTRUM documents to the following e-mail address:

spectrum-docs@aprisma.com

Thank you for helping us improve our documentation.

Online Documents

SPECTRUM documents are available online at:

http://www.aprisma.com/manuals

Check this site for the latest updates and additions.

Overview

SPECTRUM provides a generic management module that can be used to represent an SNMP-compliant network device that does not have a corresponding SPECTRUM management module. SNMP-compliant devices are supported by a series of Management Information Bases (MIBs), which are SNMP structures that describe the particular device being monitored. MIBs are imported into the SPECTRUM database and made available via device, application, and interface model types.

The generic model type, GnSNMPDev, is able to efficiently represent a broad range of devices by creating a single model to represent the device, interface models to represent the device's ports, and application models to represent each of the standard (IETF) MIBs that are supported by the device.

In many cases, the functionality provided by the GnSNMPDev management module is adequate to properly manage the device. However, there are cases where you may need to extend the capabilities of the GnSNMPDev management module to support additional MIBs or represent specialized features.

The sections in this manual provide instruction on using GnSNMPDev in various ways:

- The GnSNMPDev Management Module [Page 14]: You can use the GnSNMPDev management module to manage a standards-compliant device for which a specific management module is not available.
- Customizing the GnSNMPDev Management Module [Page 19]: You can enhance the GnSNMPDev model type by assigning a descriptive identifier based on a device's System Object Identifier, importing MIB objects from standard or proprietary MIBs and displaying them in customized views, creating customized SpectroWATCHes to monitor and analyze device statistics, and adding additional trap, event, and alarm processing.
- Developing a New Management Module [Page 30]: You can use the GnSNMPDev management module as a toolkit to derive new application and device model types. Once the new model types have been derived, you can customize how they will be displayed in the SpectroGRAPH and configure how they will process traps.

The GnSNMPDev Management Module

In This Section

Overview [Page 14]

Modeling [Page 15]

Identification [Page 15]

Interfaces [Page 16]

Applications [Page 16]

Views [Page 17]

Traps, Events and Alarms [Page 17]

Overview

The GnSNMPDev management module provides management for standards-compliant, SNMP devices for which a specific management module is not available. GnSNMPDev is an extremely powerful modeling capability as it allows SPECTRUM to dynamically create models on-the-fly to manage devices when a specific management module is unavailable.

GnSNMPDev rapidly queries the device to determine its characteristics and capabilities, then creates a model to represent the device. GnSNMPDev also creates sub-models, referred to as application models, to represent each of the standard MIBs supported by the device, and interface models to represent each device port defined in the standard MIB-II interface table. The application and interface models are associated with the GnSNMPDev model, and together they provide basic management capabilities for the device.

Devices that are modeled with the GnSNMPDev model type can be used with all SPECTRUM's management tools. Importantly, GnSNMPDev models participate fully in SPECTRUM's root cause analysis, fault isolation, and downstream alarm suppression algorithms and are thus able to alert users to network and device problems like other SPECTRUM device models.

Modeling

When modeling a device with Autodiscovery or the Model by IP Address command, SPECTRUM automatically chooses the GnSNMPDev model type when a specific management module for the device is not available. You can also choose to model a device using the GnSNMPDev model type when using the "New Model" command to model a device.

As with all management modules, you can choose to map interface model connectivity automatically using Autodiscovery or you can map this connectivity manually.

See *How to Manage Your Network with SPECTRUM (1909)* for more information on manually modeling devices and connections. See the *Autodiscovery User Guide (0727)* for more information on the Autodiscovery process.

Identification

When a device is modeled with GnSNMPDev, SPECTRUM will assign the device a descriptive identifier or device type, and an icon label which reflects the device's functionality. See the *SPECTRUM Icons (2518)* guide for more information on the icon labels that SPECTRUM uses.

To determine the model's device type, SPECTRUM will first check the System Object Identifier to Device Type mapping list (see Adding Device Type Values for Device Identification [Page 20]). If a match is found (e.g. "Cisco 2621"), it becomes the model's device type. If no match is found in the list, SPECTRUM extracts the device's enterprise ID from the System Object Identifier and uses it to determine the manufacturer. SPECTRUM then looks at the device's capabilities and based on these appends an abbreviation, i.e. Rtr, Bdg, etc., to the manufacturer name. This entire string becomes the device type name (e.g. "Cisco Rtr"). If SPECTRUM is unable to determine an appropriate device type, the default value, "SNMP DV", is assigned.

When determining the appropriate icon label for a device model, SPECTRUM determines the primary function of the device, i.e. a router, bridge, switch, workstation, etc., and assigns an icon label to the device model that represents this primary function. This icon appears in the center of the device model. Note that you can override this selection by choosing the "Device Symbol" menu selection from the GnSNMPDev model.

Interfaces

GnSNMPDev intelligence creates an interface model for every instance in the MIB-II Interface table. Interface models are instantiated and associated with the device during SPECTRUM modeling. They represent physical or logical connections on a device. The device model's Device Topology view shows all of the interfaces that SPECTRUM has discovered on a device. Interface state (`ON` or `OFF`) and status (`UP` or `DOWN`) is displayed on each interface model.

Connectivity between devices can be mapped to the port level, which gives SPECTRUM the ability to isolate faults to the same level. For example, if a port on a device goes down, an alarm will be generated on the individual interface model rather than at the device level. Interface model statistics can be polled and logged allowing you to monitor and manage the device's performance to the interface level.

Potential interface model types include: Gen_If_Port, Serial_If_Port, VLAN_IF, and FrameRelayPort.

If Frame Relay Manager is installed and the device supports either of the Frame Relay standard MIBs RFC1315 or RFC2115, the DLCI circuits will be modeled using the DLCI_port model type. See the *Frame Relay Manager User Guide (2102)* for further information.

If ATM Circuit Manager is installed and the device supports the ATM MIB RFC1695, the ATM logical connections will be modeled using the ATMVclLink or ATMVplLink model types. See the *ATM Circuit Manager User Guide (3518)* for further information.

Applications

When a device is modeled with GnSNMPDev, SPECTRUM creates application models to represent each of the standard (IETF) MIBs that the device supports. Application models are instantiated and associated with the device during SPECTRUM modeling.

For example, if GnSNMPDev intelligence detects that a modeled device performs routing functions (based on the presence of a routing MIB), a Routing Application model will be created and associated with the device model. In this manner, non-routing devices are not burdened with functionality needed to manage routers; each device model carries only the functionality it needs.

Access to all application models associated with a given device model is provided via the device's Application view. Detailed information about standard MIB applications and their views are available in the following guides: *Technology Applications (5065), Transmission Applications (5064), Routing Applications (3080), Bridging Applications (2562),* and *MIB-II Applications (2561)*.

Additional support for standard or proprietary MIBs can be added to the GnSNMPDev model type. See the section on Customizing the GnSNMPDev Management Module [Page 19] for more information.

Views

A device modeled with the GnSNMPDev model type offers access to all of SPECTRUM's generic device views, such as the Application View, Device Interface View, and Device Topology View. Detailed information on these views is available in the *SPECTRUM Views (2517)* guide.

Traps, Events and Alarms

The default trap support available with the GnSNMPDev management module is shown in Table 1. It is possible to enhance this support to include other traps and event processing. See Appendix E: Alert, Event and Alarm Processing [Page 105] for more information.

In addition, the GnSNMPDev model type supports various RFC and IEEE standard applications traps. The applications are created and associated with the device model based on the specific device's capabilities. The following guides contain complete documentation for each of the standard applications supported by SPECTRUM:

- *Bridging Applications (2562)*
- *MIB-II Applications (2561)*
- *Routing Applications (3080)*
- *Technology Applications (5065)*
- *Transmission Applications (5064)*

Table 1: Standard and Device-Specific Trap Support

Trap Name	OID	Variable Binding	Event Generated	Alarm Generated	Alarm Severity
coldStart	0.0	NA	0x10306	NA	NA
warmStart	1.0	NA	0x10307	NA	NA
linkUp	2.0	1.3.6.1.2.1.2.2.1.1 1.3.6.1.2.1.2.2.1.7 1.3.6.1.2.1.2.2.1.8	0x220001	NA	NA
linkDown	3.0	1.3.6.1.2.1.2.2.1.1 1.3.6.1.2.1.2.2.1.7 1.3.6.1.2.1.2.2.1.8	0x220002	0x220002	Yellow alarm on the device (can be configured per port) Red alarm on the Port
authenticationFailure	4.0	NA	0x1030a	0x1030a	Yellow
egpNeighborLoss	5.0	1.3.6.1.2.1.8.5.1.2	0x1030b	NA	NA

Customizing the GnSNMPDev Management Module

This section explains methods for customizing the GnSNMPDev management module in order to enhance its support for a specialized device.

In This Section

[Adding Support for Additional MIBs [Page 19]](#)

[Adding Support for Additional Alerts, Events and Alarms [Page 20]](#)

[Adding Device Type Values for Device Identification [Page 20]](#)

Adding Support for Additional MIBs

You can add support for a standard or proprietary MIB by importing MIB objects using the MIB import mechanism and the Send to SpectroSERVER command, which are both available in JMib Tools. This mechanism creates SPECTRUM attributes from these MIB objects and makes them available to all models that represent a device in the SpectroSERVER's database. See the *JMib Tools (1426)* guide for more information and instructions on using this functionality.

In order to display the MIB objects, you can use the GIB Editor to create or modify GIB views. GIB views organize and display attribute information and statistics for a specific model type. In-depth information and instructions on using the GIB Editor are available in the *GIB Editor User's Guide (0660)*.

Once you have created a GIB view to display the MIB objects, you can use conditional menu picks to make the view available to only the GnSNMPDev device models that support the MIB objects. See Implementing Conditional Menu Selections [Page 88] for complete instructions.

It is also possible to create a new application model type to support the functionality of a new MIB. You will want to do this if you also need to use the GnChassis/GnPort derivation point to create proprietary port models. For information on creating proprietary port models, see Modeling Ports and Boards [Page 61] and Appendix H: Model Fragment Reference [Page 111]. For instructions on creating an application model type, see Appendix B: Creating a New Application Model Type [Page 48].

Adding Support for Additional Alerts, Events and Alarms

The GnSNMPDev model type supports various traps, events, and alarms by default. See Traps, Events and Alarms [Page 17] for a complete listing of this support. It is possible to add support for additional SNMP traps. When you do this, you map the trap to a SPECTRUM event and then specify how you would like the event to be processed (i.e. logged, used to create an alarm, used to clear and alarm, or used as a part of an Event Rule). This enhances SPECTRUM's ability to effectively manage the changing conditions of your network. See Appendix E: Alert, Event and Alarm Processing [Page 105] and the *Event Configuration Files Guide (5070)* for more information.

Adding SpectroWATCHes to Track and Manage Model Conditions

In order to further customize a particular GnSNMPDev model, you can use the SpectroWATCH WatchEditor tool to create one or more watches for a particular model. A watch monitors and analyzes changing internal and external attribute values of a model. Watches can be built to include expressions that incorporate one or more attribute values. These attribute values, or an expression based on these values, can then be measured against a defined threshold value. Results can be used to generate events and alarms and can be logged for historical tracking and report information. Results can also be sent to script files. SPECTRUM evaluates the attribute values defined in a watch by polling those attributes. Keep in mind that this can have an impact on network traffic and system resources. Watches that are no longer useful should be deleted. For further information on creating new watches using SpectroWatch, please refer to the *SpectroWatch Operator's Reference (0919)*.

Adding Device Type Values for Device Identification

The Device Type attribute (`0x23000e`) is a text string used to identify the device being modeled. This is displayed on the lower section of the device icon. SPECTRUM provides the ability to search, filter, and report on device models using the Device Type attribute, which gives you a finer level of granularity when managing your device network infrastructure.

The Device Type Identification application allows you to assign Device Type values to GnSNMPDev models via a user-defined list of System Object Identifier to Device Type pairings. This application is run and changes take effect while the SpectroSERVER is running.

Mapping a System Object Identifier to a Device Type

The Device Type Identification application allows control of the Device Identification List, providing the ability to create new System Object Identifier to Device Type mappings. Coupled with the GnSNMPDev model type's flexibility, it is possible to model and monitor any SNMP-compliant device in the network even if a specific SPECTRUM management module is not available.

Once you have created or modified a mapping and applied the changes to the SpectroSERVER, all future device models with this System Object Identifier will be assigned the associated Device Type value. In addition, all existing device models with this System Object Identifier are updated and assigned the new Device Type value. This provides consistency when performing searches or filtering.

> **Note:** If there are device models that you wish to prevent from being modified through this procedure, refer to the Customization Considerations [Page 27] section.

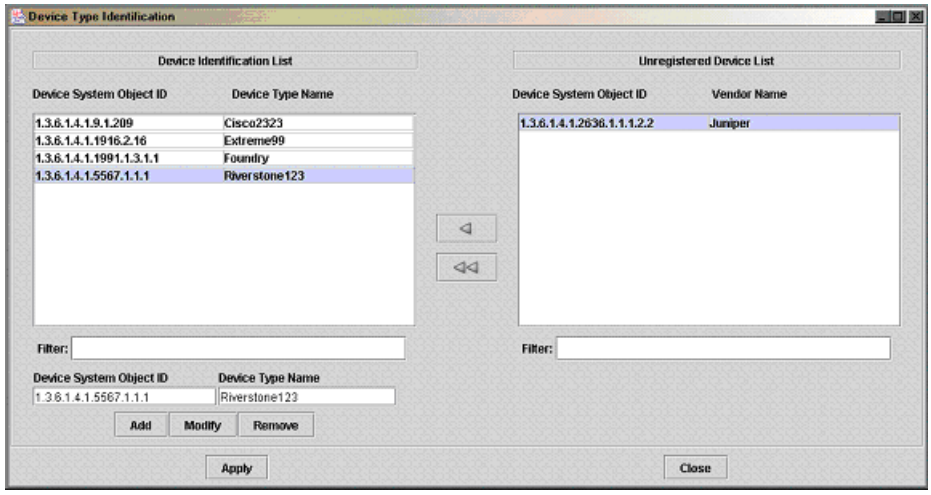
The Device Type Identification application also contains an Unregistered Device List [Page 24], which shows a list of System Object Identifiers for which a matching device type entry could not be found during Autodiscovery or Model by IP. This list allows you to search for or filter on System Object Identifiers, and then move these Identifiers to the Device Type Identification List for mapping. This can be used to facilitate the process of setting up the Device Identification list entries for all devices modeled with GnSNMPDev.

Starting the Device Type Identification Application

You must have write permissions in order to run the Device Type Identification application.

1. Open the SpectroGRAPH Topology view.
2. Select the VNM Icon, and right click to select the **Device Type Identification** command.
3. SpectroGRAPH displays the Device Type Identification window (Figure 1).

Figure 1: The Device Type Identification Application



The panel on the left shows the The Device Identification List [Page 22], which displays the current System Object Identifier to Device Type mappings. The panel on the right shows the Unregistered Device List [Page 24], which displays all System Object Identifiers and their corresponding vendor names for which a matching device type entry could not be found.

The Device Identification List

The Device Identification List (located on the left hand side of the application) shows a list of System Object Identifiers and their matching Device Type Name.

To search this list for specific values, you can filter on System Object Identifier or Device Type Name.

Creating a New Entry

1. Type a device's System Object Identifier in the **Device System Object ID** text field, for example, `1.3.6.1.4.1.9.5.60`.
2. Type the device type name you would like to pair with this System Object Identifier in the **Device Type Name** text field, for example `Catalyst 7613`.
3. Click the **Add** button to add the entry to the Device Identification List.

Modifying an Existing Entry

1. Select the entry to be modified from the Device Identification List. Notice the selection fills in the **Device System Object ID** and **Device Type Name** text fields.
2. Modify the System Object Identifier in the **Device System Object ID** text field to the desired value, for example, `1.3.6.1.4.1.9.5.55`.
3. Modify the device type in the **Device Type Name** text field to the desired value, for example `Catalyst 7609`.
4. Click the **Modify** button to complete the modification of the entry.

Removing an Existing Entry

1. Select the entry to be deleted from the Device Identification List. Notice the selection fills in the **Device System Object ID** and **Device Type Name** text fields.
2. Click the **Remove** button to remove the entry.

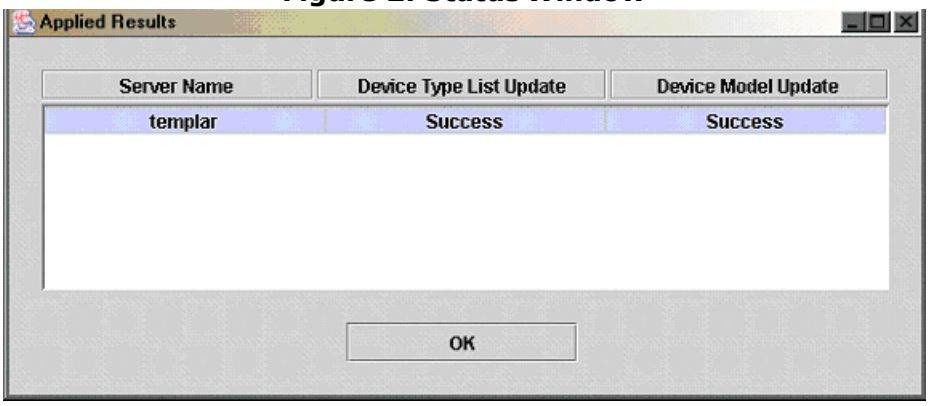
Applying Changes to the SpectroSERVER

1. Once you have completed any changes or additions to the list, click the **Apply** button.
2. All device models whose System Object Identifiers are on the Device Identification List and whose `Enable_IH_Device_Name` attribute is set to `TRUE` will be updated. The specified device type names will appear on the respective device models' icons.

   The `Enable_IH_Device_Name` attribute allows you to control whether or not customizations to the Device Type can be overwritten. Setting a model's `Enable_IH_Device_Name` attribute value to `FALSE` prevents the model's Device Type from being changed. See Customization Considerations [Page 27] for more instructions on setting this attribute.
3. A window will appear displaying the status of these changes (Figure 2).

   Note: System Object Identifier to Device Type pairings are preserved regardless of Service Pack upgrades or database migrations.

Figure 2: Status Window



Unregistered Device List

The Unregistered Device List (located on the right hand side of the application) shows a list of System Object Identifiers for models of the GnSNMPDev model type which a matching device type entry could not be found during Autodiscovery or Model by IP. This panel contains two columns: the System Object Identifier and the associated vendor name. If a vendor cannot be identified, the value "SNMP DV" is associated with the System Object Identifier.

You can move entries from this list to the Device Identification List in order to add the appropriate Device Type. To search this list for specific values, you can filter on System Object Identifier or vendor.

Moving a single unregistered device list entry

1. Select the appropriate entry from the Unregistered Device List.
2. Select the left single arrow button. The entry will move from the Unregistered Device List to the Device Identification List.
3. Modify the entry's Device Type value as needed.

Moving all unregistered device list entries

1. Select the left double arrow button. All entries are moved from the Unregistered Device List to the Device Identification List.
2. Modify the entries' Device Type value.

Setting up Device Identification List entries for all GnSNMPDev models

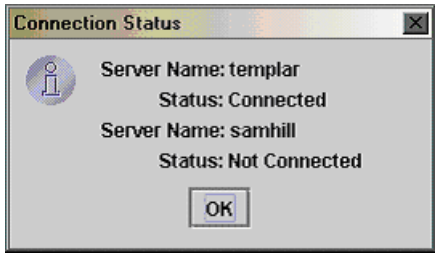
The Unregistered Device List can be used to simplify the initial task of setting up Device Identification list entries for all devices modeled with GnSNMPDev. Instead of trying to determine all of the devices that use the GnSNMPDev model type and then creating entries in the Device Identification List for each, first model the devices. Once modeled, those that do not contain an entry in the Device Identification List will be added to the Unregistered Device List. Once the device modeling is complete, launch the Device Type Identification application. Move unregistered device entries to the Device Identification List, modify the Device Type value for each entry, and apply the change. All current device models will be updated and future device models will use the new Device Type.

Device Type Identification in a Distributed Server Environment

The Device Type Identification application provides support for customers that have multiple SpectroSERVERs deployed in a distributed environment. When launched from the SpectroGRAPH, the application will identify all SpectroSERVERs in the distributed environment.

If contact to a server within the distributed environment cannot be established, a window will appear showing the connection status to each server.

Figure 3: Connection Status for Device Type Identification

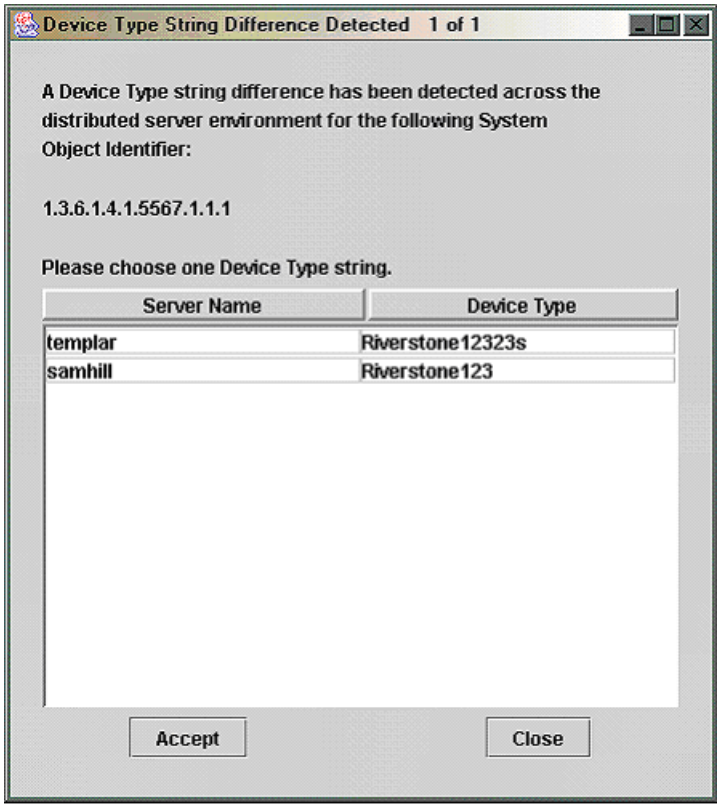


Note: Problems contacting SpectroSERVERs in your distributed environment may be due to SpectroSERVERs belonging to a broadcast domain unknown to the Visibroker Smart Agent running on the local machine. Refer to the "Configuring SPECTRUM CORBA Services" section of the *Installation Guide (5068)* to fix this problem.

Once the Device Type Identification application has found all of the servers in the distributed environment, it will obtain a Device Identification List from each. It will identify any inconsistencies amongst the Device Type System Object Identifier mappings from the collected lists. When an inconsistency is identified, a window will appear displaying the inconsistent entries (see Figure 4). Resolve the inconsistency by choosing the entry you wish to use.

For example, the window in Figure 4 would appear if the server "templar" contained a mapping of `1.3.6.1.4.1.5567.1.1.1` to `Riverstone12323s` and the server "samhill" contained a mapping of the same System Object ID `1.3.6.1.4.1.5567.1.1.1` to `Riverstone123`.

Figure 4: Device Type String Difference Detected



If the Device Type Identification application finds a Device Type entry for a System Object Identifier that is present on some but not all of the servers

in the distributed environment, this entry will be the selected choice for all servers.

In a distributed environment, any changes made to the Device Identification List will be applied to all servers. Each server will report back a status regarding when the change is applied.

Working in a Fault Tolerant Environment

If you are working in a fault tolerant environment, the Device Identification application has the ability to differentiate between a primary and a backup server. The application will connect and work with the primary server. The application will not connect to the backup server, but the backup server will obtain the Device Identifier List update and device model updates during the Online Backup procedure.

WARNING! Do not make changes with the Device Identification application on the backup server. Once the primary server is up and running again, these changes will not be propagated to the primary server.

See the *Distributed SpectroSERVER (2770)* guide for more information on working in a fault tolerant environment.

Customization Considerations

It is possible to set different Device Type values for device models with the same System Object Identifier through the use of the Global Attribute Editor. However, when making future modifications to Device Type values, it is possible for such customizations to be overwritten. To avoid this, you can use the `Enable_IH_Device_Name` attribute (`0x3d0008`). Consider the following scenario:

1. The following new System Object ID-Device Type entry has been added:

System Object ID	Device Type
1.3.6.1.4.1.311.1	PC

2. After applying the changes, 2 devices with the System Object Identifier 1.3.6.1.4.1.311.1 modeled with the GnSNMPDev model type will receive the new Device Type.

Model Name	Device Type
Model1	PC
Model2	PC

3. Using the Search Manager, perform a search on the Device Type "PC" and find both models. Using the Global Attribute Editor, change the device type value for Model1 from "PC" to "MailServerPC."
4. Later you decide that the System Object Identifier 1.3.6.1.4.1.311.1 needs a more descriptive Device Type, and use the Device Type Identification application to change it from "PC" to "WindowsNT".
5. Once you apply the changes made in Step 4, all GnSNMPDev models with that System Object Identifier will be changed, including the one modified in Step 3.
6. To prevent the customization made in Step 3 from being overwritten, you can set the `Enable_IH_Device_Name` attribute (`0x3d0008`) on the device model to `FALSE` using the following procedure:
   a. From a SpectroGRAPH view, select the **Tools** drop down menu and select **Search Manager**. This will launch the Search Manager application.
   b. From the Search Manager application, search for the device models that have been customized.
   c. Highlight the customized device models and select the **Management** drop down menu. From this menu select **Set Attribute Values...**. This will launch the Global Attribute Editor view.
   d. From the Global Attribute Editor view, select the **User Defined** tab. In this tab, select the **Add** button and the Attribute Selector view will appear.
   e. In the **Filter** text field, enter `Enable_IH_Device_Name`.
   f. Click the **OK** button. Attribute `Enable_IH_Device_Name` will appear in the User Defined tab.
   g. Select value **False** and click the **Apply** button.

See the *Search Manager User Guide (2383)* for instructions on using the Search Manager and the Global Attribute Editor.

Developing a New Management Module

If you determine that the customization options outlined in the preceding chapter do not meet your device management needs, consider creating a new device model type or a new application model type using GnSNMPDev. The following sections discuss some common scenarios in which this would be required.

In This Section

Representing the Device with Model Types [Page 30]

Additional MIB Support [Page 31]

Unique Device Model Icon and Symbol [Page 31]

Unique SpectroWATCH [Page 32]

Device Identification [Page 32]

Unique Trap Mapping [Page 33]

Interface/Port Model Creation [Page 33]

Representing the Device with Model Types

A device model type and its associated interface and application model types collectively model a device. When you consider creating a new device model type or simply adding to the functionality of the GnSNMPDev device model type, you must understand the functional components of the device in order to effectively expand SPECTRUM's management of the device.

There are many device functions that are supported by both the GnSNMPDev device model type and the interface and application models known to the GnSNMPDev model type. These include functionality from both proprietary and standard MIBs. Before creating a new device or application model type, it is best to identify the functionality of the device that is already supported by GnSNMPDev. For example, if a device's interfaces map one to one with physical ports on a single board, GnSNMPDev can support this device, without enhancement, via its built-in support for MIB-II interfaces in the Snmp2_Agent application model.

To find out how GnSNMPDev will support the device, create a model of the device using the Model by IP command in SpectroGRAPH. SPECTRUM automatically finds the model type most appropriate for the device. If there is not a specific model type, SPECTRUM will choose the GnSNMPDev model type, and instantiate a GnSNMPDev model to represent the device.

Once you have established the type of support SPECTRUM will provide for your device by default and considered the customizations described in Customizing the GnSNMPDev Management Module [Page 19], you can decide if further customization is necessary in order to manage your device properly. The following sections outline some scenarios in which expanded support might be necessary.

Additional MIB Support

If access to additional MIBs is required for your device management needs, there are a number of ways in which MIB objects can be made available to a device model:

- The first and recommended method is to import the new MIB directly into the SpectroSERVER using the import mechanism provided in JMib Tools. For further information on this method see Adding Support for Additional MIBs [Page 19] and the *JMib Tools (1426)* guide.
- The second method is to create a new device model type to represent your device and derive the necessary MIBs into the device model type. See Appendix A: Creating a New Device Model Type [Page 34] for complete instructions.
- The third method is to create a new application model type that provides access to the new MIB. See Appendix B: Creating a New Application Model Type [Page 48] for complete instructions.

Unique Device Model Icon and Symbol

If you want to represent a device with a unique device model icon or icon symbol, you must create a new model type to represent your device. See Appendix A: Creating a New Device Model Type [Page 34] for complete instructions on creating a new device model type and Appendix D: Customizing Views and Device Models [Page 67] for details on how to create a unique device model icon or icon symbol.

Unique SpectroWATCH

The GnSNMPDev model type provides a number of predefined SpectroWATCHes. These watches can be turned on and off on a per model basis. If you need to customize the SpectroWATCH implementation on the models that represent your device, you can do this for each applicable GnSNMPDev model that is instantiated. To avoid having to repeat this customization on each model, you can create your own device model type to implement the customized SpectroWATCH configuration.

For more information, see Appendix A: Creating a New Device Model Type [Page 34] and Appendix F: SpectroWATCH [Page 108].

Device Identification

In some cases SPECTRUM may not be able to uniquely identify your device due to the lack of a unique sysObjectID or other unique variable. If it is essential to be able to uniquely identify this device in the SpectroGRAPH Topology views using the Device Type attribute, one of the following two methods can be used.

- The first option is to derive a new device model type and set the Device Type attribute directly in the database as a default value. To model your device, use SPECTRUM's New Model command, which allows you to choose the model type used to model the device. See Appendix A: Creating a New Device Model Type [Page 34] for complete instructions on creating a new device model type.
- The second option is to create a new application model type that represents a unique aspect of your device, and then provide a menu pick that is enabled when this application model type is supported. In this way, identification is provided via the presence of the menu pick.

  For example, SPECTRUM implements this method of identification for devices running the Cisco CallManager software. If SPECTRUM detects such a device, it instantiates a CallManager application model for the device model and enables the "Cisco CallManager" menu selection that accesses information provided by the Call Manager software. See the *Cisco CallManager (5116)* guide for more information on this solution.

  See Appendix B: Creating a New Application Model Type [Page 48] and Implementing Conditional Menu Selections [Page 88] for more information.

Unique Trap Mapping

You will need to create a new device model type if the device you are modeling requires that you implement unique trap processing in response to a common trap. For example, SPECTRUM supports three variable bindings sent with the communications link down trap: `IfIndex`, `OperStatus`, and `AdminStatus`. If the device you are supporting sends additional variable bindings with this trap, by default the information in these variable bindings would not be passed to SPECTRUM. Thus the data from these variable bindings would not be available in the Event Format file.

To work around this, you must create a new device model type and create support for this trap in the device model type's AlertMap, EventDisp, and Event Format files. This support would override SPECTRUM's default processing for the communications link down trap for this model type only.

See Appendix A: Creating a New Device Model Type [Page 34] for instructions on creating a new device model type and Appendix E: Alert, Event and Alarm Processing [Page 105] for information on trap processing.

Interface/Port Model Creation

If your device does not advertise its interface or port information in MIB-II's standard interface table, but instead uses information from a proprietary MIB, SPECTRUM will not be able to model the ports on your device. Without port models you will be unable to resolve connections to the port level and you will be unable to monitor the status of each port. To work around this, you must create a new application model type that includes support for the proprietary MIB with the interface or port information. See Appendix B: Creating a New Application Model Type [Page 48] and Interface/Port Model Creation [Page 33] for instructions.

Appendix A: Creating a New Device Model Type

This section describes the factors to consider and steps to perform if you choose to create a new device model type. These include understanding database derivation and MIB requirements, creating the model type and setting required attributes, selecting discovery and identification mechanisms, creating the SpectroGRAPH support files, making desired customizations, and making the new model type distributable to other SPECTRUM hosts.

In This Section

Designing a New Device Model Type [Page 34]

Creating a New Device Model Type [Page 37]

Configuring a New Device Model Type [Page 38]

Creating SpectroGRAPH Support Files for a New Device Model Type [Page 46]

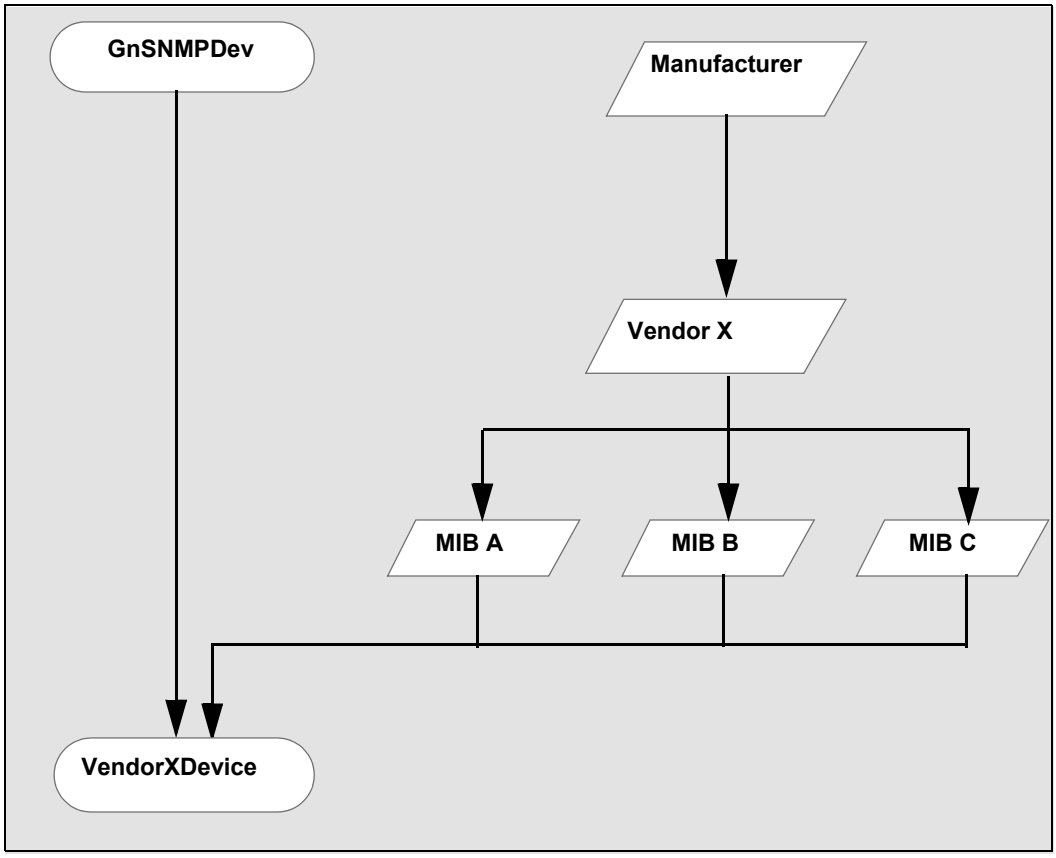
Customizing a New Device Model Type [Page 46]

Distributing a New Device Model Type [Page 47]

Designing a New Device Model Type

The device model type database architecture used for developing new device model types is an organized structure where all proprietary MIBs are imported into separate model types and derived directly into the device model type as shown in Figure 5.

Figure 5: Aprisma Device Model Type Architecture



This scheme has many advantages:

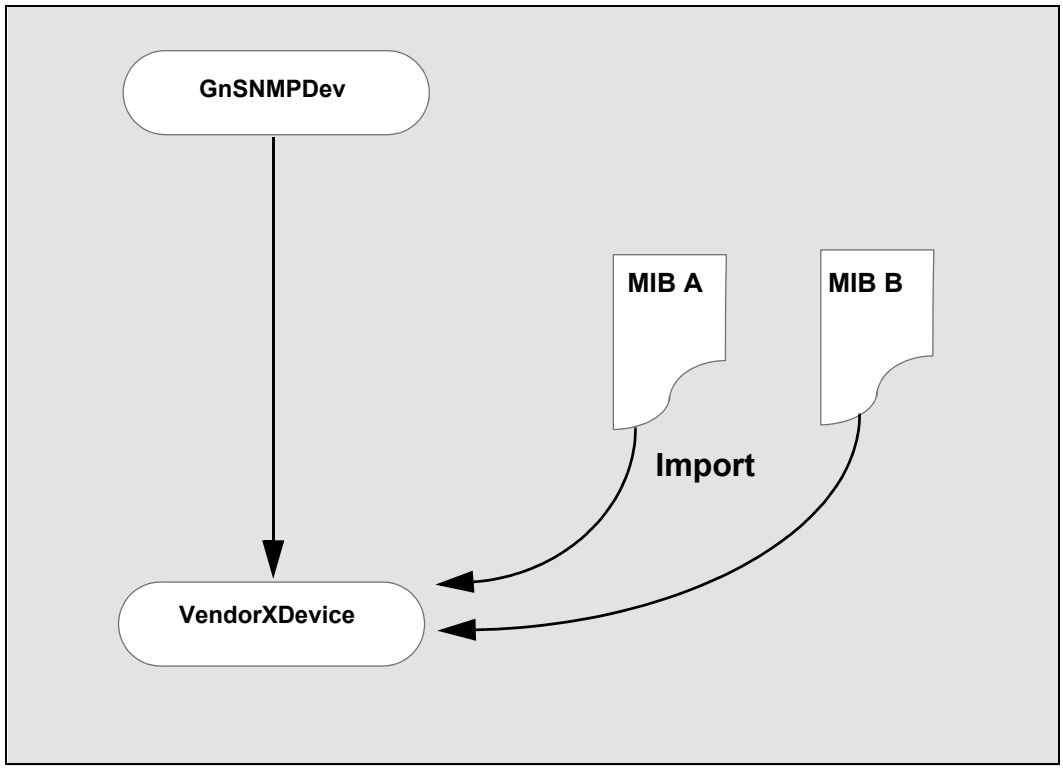
- It allows a single MIB to be derived into multiple model types while maintaining the attribute IDs.
- It allows vendor MIBs to be organized in a single collection.
- It allows for convenient access to MIB information directly from the device model. For example, GIB views, SpectroWatches, logging and polling of proprietary vendor attributes can all be accessed from the device model.

If there are additional MIBs that your new device model requires access to from SPECTRUM, the simplest method to use is the MIB import mechanism and the Send to SpectroSERVER command available in JMibTools. This mechanism creates attributes from these objects and makes them available to all models that represent a device in the SpectroSERVER's database. See the *JMib Tools (1426)* guide for more information and instructions on using this functionality. Note also that the MIB import

mechanism will distribute the new MIB across all SpectroSERVERS in a distributed environment.

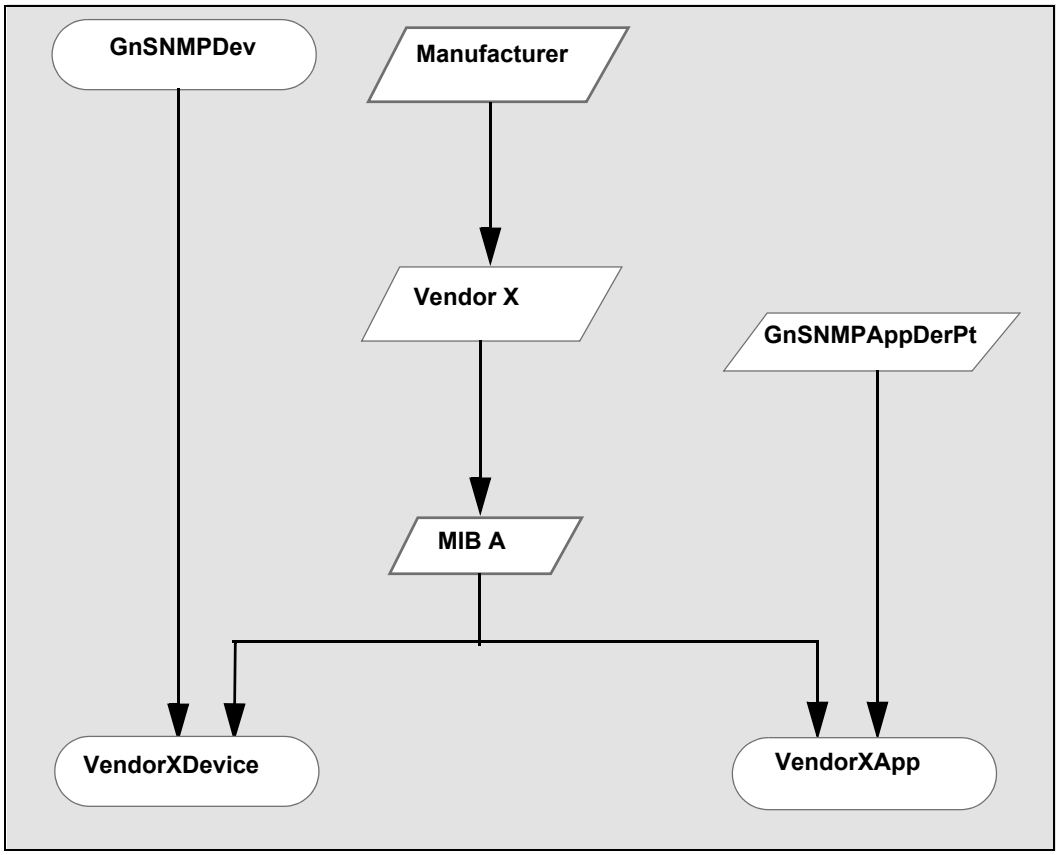
However, if you are required to install the new device model type on SpectroSERVERs that are not in a distributed environment, then you may want to import the MIBs directly into a new device model type and create a virtual CD (VCD) which contains all components of the new device model type. The recommended device model type database architecture is one where all proprietary MIBs are imported directly into the device model type as shown in Figure 6. This is a slightly simplified version of the architecture depicted in Figure 5.

Figure 6: Recommended Device Model Type Architecture



An alternate architecture may be used if a single MIB will be derived into multiple model types, either multiple device model types or perhaps a device and application model type. In this scenario, it may be advantageous to derive the MIB into a separate model type that can be used as a derivation point, thus allowing the attribute IDs to be maintained across the derived model types. Organizing this MIB model type under a new or existing vendor model type helps keep the database organized, as shown in Figure 7.

Figure 7: Alternate Device Model Type Architecture



Creating a New Device Model Type

Once you have determined the desired database scheme, use the Model Type Editor (MTE) to create your model types. All device model types should be derived from the GnSNMPDev model type. For step-by-step examples of creating a new model type, see Appendix J: Tutorials [Page 153]. For instructions on using the MTE, see the *Model Type Editor User Guide (0659)*.

When you are using the MTE to create a new device model type, it is important to remember to correctly set the model type flags for the new model type. Instructions on setting these flags are given in the following section.

Model Type Flags

It is important to set the value of various model type flags to ensure that models of this model type behave properly within SPECTRUM. Each flag represents a Boolean value and can either be selected (set to `TRUE`) or deselected (set to `FALSE`).

In most cases you should select (set to `TRUE`) the `Visible`, `Instantiable`, and `Derivable` flags.

- If the `Visible` flag is set to `TRUE`, the model type is visible to all MTE users. If the `Visible` flag is set to `FALSE`, the model type is only viewable to a user with the same developer ID as the one used to create the model type.
- If the `Instantiable` flag is set to `TRUE`, you can instantiate a model of this model type in SpectroGRAPH.
- If the `Derivable` flag is set to `TRUE`, this model type can be used as a base for other model types.

In most cases you should deselect (set to `FALSE`) the `No Destroy`, `Unique`, and `Required` flags.

- If the `No Destroy` flag is set to `TRUE`, users are not able to destroy a model of this type in SpectroGRAPH.
- If the `Unique` flag is set to `TRUE`, only one model of this model type can be instantiated in SpectroGRAPH.
- If the `Required` flag is set to `TRUE`, then a model of this model type must always exist in the SpectroSERVER database.

Configuring a New Device Model Type

There are a few steps involved in configuring a new device model type. They include setting attribute values, configuring SPECTRUM to select the appropriate device model type and device identification, and configuring serial number handling. The following sections give instructions on each of these configuration steps.

Setting Attribute Values

Once you have created your new device model type, you will need to use MTE to set the default value of several attributes. Some of these settings

are used to configure various built-in capabilities inherited by deriving from the GnSNMPDev model type. Table 2 describes the attributes and settings.

Table 2: Attribute Values

Attribute Name	Description
CompanyName	The name of the company developing the management module.
Description	There are two description attributes, one in the `MMDeveloper` group and one in the `CommonInfo` group. The `Description` attribute in the `MMDeveloper` group has a default value of Generic SNMP Device Management Module. It is recommended that you reset this default value with a basic description of your management module. The `Description` attribute in the `CommonInfo` group can be filled in with the identical text or left empty.
Desc_Key_Word	In the case that `System_Desc_Verify` or `Vendor_Object_ID` discovery mechanisms identify multiple device model types, `sysDescr` can be searched for a substring match from the value of this attribute.
DeviceNameList	Used in conjunction with `SysOIDVerifyList`. Populate this attribute with text strings describing the device. These will be matched up against the entries found in `SysOIDVerifyList`. Device discovery will run through the list of `sysObjectIDs` and if a match is found, the coinciding text string entry will be populated into the `DeviceType` attribute.
DeviceSerialAttr	Set this to the attribute ID of the external attribute which contains the serial number of the device. When the model is created, it will read this external attribute and write it into `Serial_Number`.

DeviceType	A description that identifies the device. A default value is required for this attribute when the `DeviceNameList` identification mechanism is not used. By setting the default value, this will guarantee a value will be present for displaying, sorting and filtering.
disposable_precedence	This attribute is evaluated at device model type discovery time when more than one model type is identified as a possible candidate. The higher value will be the chosen model type. This value is also evaluated when a model is created that has the same MAC address as a previously existing model, then both of those model's `disposable_precedence` attributes are evaluated. The model with the higher value will replace the existing model by stealing its `CONNECTs` associations.
Enable_IH_Device_Name	This attribute enables generic device type naming. Typically, this mechanism is used by GnSNMPDev. This should be set to `FALSE` when deriving a new device model type.
Enable_IH_Spec_Dev_Name	This attribute enables the specific device model type intelligence which uses `SysOIDVerifyList` and `DeviceNameList`. This attribute should only be set to `TRUE` when using this mechanism. Otherwise leave it set to `FALSE`.
Manufacturer	Name of the vendor that manufactures the device.
MMName	The name of the management module.
MMPartNumber	The part number you will be giving to the management module.
System_Desc_Verify	This provides a device model type discovery mechanism by which the `sysDescr` is parsed for firmware version information. Clear this default value when not using as it will cause problems for the other discovery methods if set.
System_Oid_Verify	This is a legacy attribute. Please refer to `SysOIDVerifyList`.

SysOIDVerifyList	Populating this attribute list with `sysObjectID` values allows device model type discovery intelligence to match the list against the device's `sysObjectID` value. If a match occurs then this model type will be chosen as a possible contender to be used for modeling.
Vendor_Name	The name of the company developing the management module.
Vendor_Object_ID	This provides a device model type discovery mechanism by which a partial `sysObjectID` match identifies a device model type.
Verify_Mismatch_Model	Set to `TRUE`. This attribute will cause SPECTRUM to perform checks for a device model type match with the device being modeled.
WebAdminURL	This attribute allows both the SpectroGRAPH and PGUI icons to launch the URL web interface for the device. Attribute IDs can be used within angled brackets and the icons are smart enough to replace the attribute ID with value of the attribute. See the *Search Manager User Guide (2383)* for a complete description of the WebAdminURL attribute.

Ensuring SPECTRUM Selects the Appropriate Device Model Type and Provides Identification

It is important to uniquely identify a device on the network. Most commonly this is done through the MIB-II object `sysObjectID`. Most vendors will assign a unique `sysObjectID` value for a particular device, thus creating a one-to-one mapping. Many vendors advertise this information in a Products MIB, which is an excellent source to obtain the mapping of `sysObjectID` to device type.

If your device provides a unique `sysObjectID` value, then use the SysOIDVerifyList/DeviceNameList Discovery and Identification Mechanism [Page 42] described below to uniquely identify your device.

If your device does not provide a unique `sysObjectID`, but does provide a unique substring within the `sysDescr`, then refer to the Vendor_Object_ID/

Desc_Key_Word Discovery and Identification Mechanism [Page 43] described below to uniquely identify your device.

If your device does not provide a unique `sysObjectID`, but does provide a firmware version text string in the `sysDescr`, refer to the System_Desc_Verify/Desc_Key_Word Discovery and Identification Mechanism [Page 44] described below to uniquely identify your device.

SysOIDVerifyList/DeviceNameList Discovery and Identification Mechanism

If your device has a unique `sysObjectID` value, you must relate the model type to the `sysObjectID` to ensure that SPECTRUM selects the new device model type to represent the device. To do this, add the `sysObjectID` value to the `SysOIDVerifyList` model type attribute. If the new device model type represents a family of devices, then add each `sysObjectID` value.

> **Note:** If another model type contains the same `sysObjectID` value in its `SysOIDVerifyList` attribute, it is possible that SPECTRUM will choose the other model type to represent a device with this `sysObjectID`. If this occurs, you should change the `disposable_precedence` attribute value on your device model type to higher value than that of the other model type. For example, if the other model type has a `disposable_precedence` value of 10, change the `disposable_precedence` value on your model type to 11.

In order to provide identification to your model, you can configure the model type to display a different device name for each of the devices that the model type is designed to support. For example, assume your device model type represents the 8480 series of switches made by MySwitch Inc. Instead of seeing the device name `MySwitch_8480XX` for all of the switches in the 8480 family, you would like to display the model number of the switch as appropriate. If SPECTRUM is modeling an 8480-09 switch, the model should display the device name `MySwitch_8480-09`. If SPECTRUM is modeling an 06 switch, the model should display the device name `MySwitch_8480-06`.

The following steps show how to set up each of the relevant attributes to enable this functionality on the model type.

1. Set the `Enable_IH_Spec_Dev_Name` attribute to `TRUE`. This attribute allows the exact device name to be determined.

2. Set the `Enable_IH_Device_Name` attribute to `FALSE`. This attribute allows the exact vendor name to be determined for GnSNMPDev model types.
3. Set the `SysOIDVerifyList` attribute equal to the `sysObjectID`(s) of the devices that the model type will represent.
4. Set the `DeviceNameList` attribute equal to the device names that apply to each `sysObjectID` listed in the `SysOIDVerifyList` attribute. Specify the same number of names in the `DeviceNameList` attribute as there are `sysObjectIDs` listed in the `SysOIDVerifyList` attribute. The names should be listed in the same order as their corresponding `sysObjectIDs`.
5. Clear the `System_Desc_Verify` default value.

   Note: The `DeviceNameList` attribute will only work for device model types that use the `SysOIDVerifyList` attribute model type discovery mechanism. Make sure that both lists have the same number of entries. Otherwise, the DeviceType will not be set correctly.

Vendor_Object_ID/Desc_Key_Word Discovery and Identification Mechanism

If your device does not provide a unique `sysObjectID`, a partial or complete match of the `sysObjectID` in combination with a `sysDescr` substring may provide unique identification.

The following steps show you how to set up each of the relevant attributes to enable this functionality.

1. Set the `Vendor_Object_ID` attribute equal to the partial or complete value of `sysObjectID` returned by your device. Note that only the first 7 terms (up to the enterprise ID) will be used for comparison.
2. Set the `Desc_Key_Word` attribute equal to the unique partial value of `sysDescr` returned by your device.
3. Set the `Enable_IH_Device_Name` attribute to `FALSE`. This attribute allows the exact vendor name to be determined.
4. Set the DeviceType attribute equal to the desired identification string.
5. Clear the `System_Desc_Verify` default value.

System_Desc_Verify/Desc_Key_Word Discovery and Identification Mechanism

If your device does not provide a unique `sysObjectID` or a unique sub-string within the `sysDescr`, check if the `sysDescr` provides a unique firmware version. This discovery mechanism searches the `sysDescr` value for either "`Version`" or "`Revi`". If one of these strings is found, the value of `System_Desc_Verify` is compared against the text that follows these key words. If a match is found, the device model type will be chosen. In the case where multiple model types have the same `System_Desc_Verify` value, a substring in the `sysDescr` can be compared by setting the `Desc_Key_Word`.

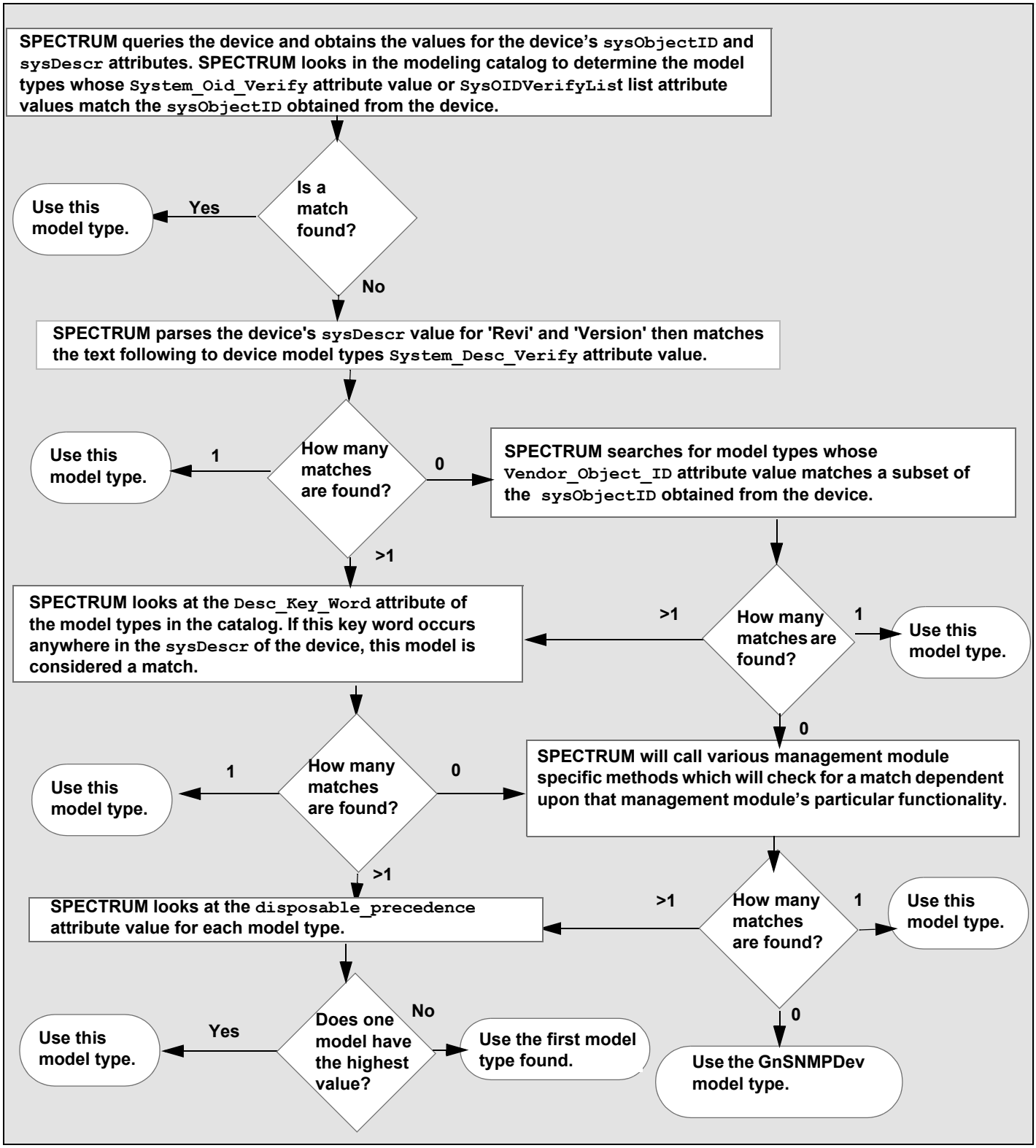
The following steps show you how to set up each of the relevant attributes to enable this functionality.

1. Set the `System_Desc_Verify` attribute equal to the contents of `sysDescr` that follows the key text noted above.
2. Set the `Desc_Key_Word` attribute equal to the unique partial value of `sysDescr` returned by your device.
3. Set the `Enable_IH_Device_Name` attribute to `FALSE`. This attribute allows the exact vendor name to be determined.
4. Set the `DeviceType` attribute equal to the desired identification string.

Discovery and Identification Flowchart

The above discovery and identification mechanisms are illustrated in the flowchart shown in Figure 8. These are the steps that SPECTRUM takes to determine which device model type should be chosen to represent a specific device.

Figure 8: How SPECTRUM Selects a Device Model Type


SPECTRUM queries the device and obtains the values for the device's sysObjectID and sysDescr attributes. SPECTRUM looks in the modeling catalog to determine the model types whose System_Oid_Verify attribute value or SysOIDVerifyList list attribute values match the sysObjectID obtained from the device.
Is a match found?
Yes
Use this model type.
No
SPECTRUM parses the device's sysDescr value for 'Revi' and 'Version' then matches the text following to device model types System_Desc_Verify attribute value.
How many matches are found?
1
Use this model type.
0
SPECTRUM searches for model types whose Vendor_Object_ID attribute value matches a subset of the sysObjectID obtained from the device.
>1
SPECTRUM looks at the Desc_Key_Word attribute of the model types in the catalog. If this key word occurs anywhere in the sysDescr of the device, this model is considered a match.
How many matches are found?
>1
1
Use this model type.
0
How many matches are found?
1
Use this model type.
0
SPECTRUM will call various management module specific methods which will check for a match dependent upon that management module's particular functionality.
>1
SPECTRUM looks at the disposable_precedence attribute value for each model type.
How many matches are found?
>1
1
Use this model type.
0
Does one model have the highest value?
Yes
Use this model type.
No
Use the first model type found.
Use the GnSNMPDev model type.

Configuring Serial Number Handling

Device model types contain an attribute used for setting and displaying the serial number of the modeled device: `Serial_Number (0x10030)`. You can enter the appropriate serial number value in any of the views where the attribute is displayed, or if the serial number is available as an external attribute from the device, you can configure the model type to automatically retrieve this value and set the `Serial_Number` attribute. To do this:

1. Ensure that the external attribute that contains the serial number is not a list attribute and is of type `TEXT_STRING or OCTET_STRING`.
2. Use the Model Type Editor to set the value of the device model type's `DeviceSerialAttr` (`0x3d0063`) attribute equal to the ID of the external attribute that contains the serial number.

When a model of this model type is instantiated, SPECTRUM will set the `Serial_Number` attribute equal to the value of the external attribute containing the serial number.

Saving Your Changes

Once you have completed deriving your new device model type in the Model Type Editor, you should save all changes in the Attributes view and then save the changes to the permanent catalog before exiting the Model Type Editor.

Creating SpectroGRAPH Support Files for a New Device Model Type

The `mmbuild` tool lets you to create SpectroGRAPH support files for a model type. This tool incorporates the new model type into the database, and links it to all the SpectroGRAPH support files required to represent it in the SpectroGRAPH. The `mmbuild` tool also links the new model type to the files that allow the icons representing it to appear in the other SPECTRUM views. For more information, refer to Appendix C: Creating SpectroGRAPH Support Files for New Model Types [Page 64].

Customizing a New Device Model Type

Although it is not necessary for the operation of the model type, once the basic support files have been created, you can customize the appearance

of the device model displayed in the SpectroGRAPH, Alarm Manager or Search Manager. You can also customize or add to the views that display information about the device model type. There are a few different types of generic views that can be edited using the GIB Editor tool. The generic views include Application views, Configuration views, Information views, and Performance views. These views display attribute and statistical information about a specific model. You can customize these views to include additional attributes, graphs, gauges, pie charts, navigational buttons, and textual annotations. It is also possible to create a new generic view. Once a generic view has been customized or created, the changes in that view are available to all models of that model type. For more information and instructions, see Appendix D: Customizing Views and Device Models [Page 67].

Distributing a New Device Model Type

The SPECTRUM Extension Integration (SEI) toolkit allows you to create a virtual CD (VCD). The VCD enables you to distribute new model types to other SPECTRUM hosts. For more information, see Appendix G: Distribution [Page 109].

Appendix B: Creating a New Application Model Type

This section describes how to expand support for a device using application model types. All application model types are derived from a series of standard model types, or derivation points. An Application will often correspond to the functionality of a MIB.

In This Section

Understanding Derivation Points and Model Fragments [Page 48]

Choosing a Derivation Point [Page 51]

Board and Port Considerations [Page 52]

Creating Application Model Types [Page 55]

Modeling Ports and Boards [Page 61]

Creating SpectroGRAPH Support Files for a New Application Model Type [Page 63]

Distribution [Page 63]

Understanding Derivation Points and Model Fragments

Derivation points must be selected and used as base model types for new application model types. All of these derivation points have functionality designed to support different types of applications. When you derive a new model type from one or more of these derivation points, the model type will inherit the functionality of those derivation points.

Some derivation points require the use of model fragments. The model fragments that are available are model types that have specific inference handlers attached to them. These inference handlers provide the model fragments with certain capabilities such as the ability to create port or board models. In order to use the functionality provided by these inference handlers, you must map attribute IDs from the model type representing the MIB to specific model fragment attribute values.

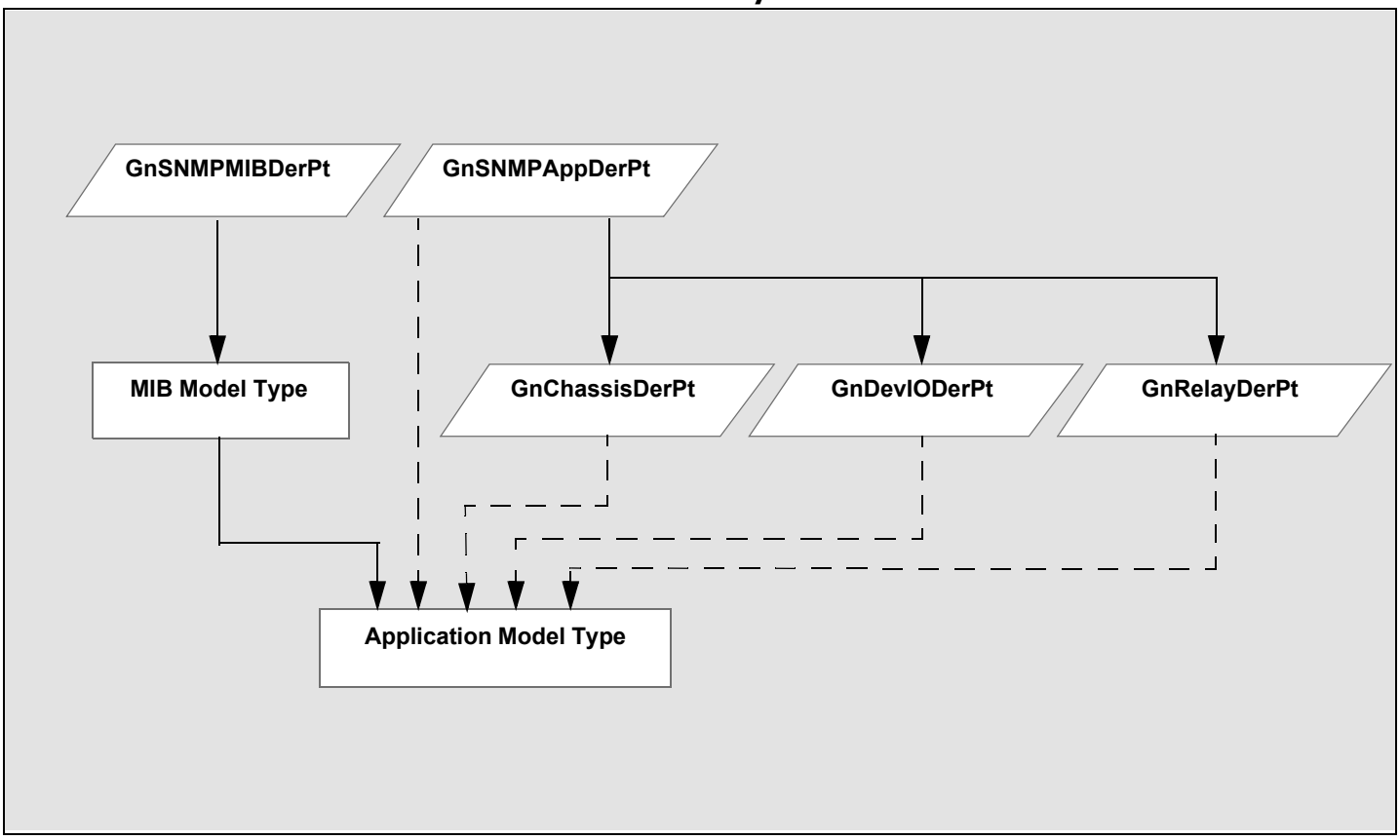
Usually, model fragments are included as base model types for the GnSNMPDev derivation points that require them. However, you may find it necessary to add a model fragment as a base model type to your new model type to take advantage of the capabilities of the inference handler attached to the model fragment.

The following model types can be used as application derivation points:

- GnSNMPMibDerPt
- GnSNMPAppDerPt
- GnChassisDerPt
- GnDevIODerPt
- GnRelayDerPt

Figure 9 shows the application derivation point hierarchy and sample derived model types. The lines connecting the model types denote the inheritance structure. Note that you will want to select only one of the dotted line paths for you new application model type derivation hierarchy.

Figure 9: Application Derivation Point Hierarchy



The derivation points for application model types are all are designed to provide you with specific functionality. `GnChassisDerPt`, `GnDevIODerPt`, and `GnRelayDerPt` each have model fragments that enhance this functionality. Table 3 shows each derivation point, the application model type it is used to create, and its associated model fragments. For a definition of each model fragment, see the Appendix H: Model Fragment Reference [Page 111].

Table 3: Derivation Points

Derivation Point	Model Type	Associated Model Fragment
GnSNMPMibDerPt	MIB Model Type	N/A
GnSNMPAppDerPt	Application model type that does not need to manage ports or boards.	N/A
GnChassisDerPt	Application model type specifically designed to model chassis functionality. It provides management for ports and boards.	GnChassis_MF
GnDevIODerPt	Application model type specifically for devices that need port management, but not board management (switches, terminal servers, etc.).	GnDeviceIO_MF
GnRelayDerPt	Application model type specifically for repeater functionality	GnDataRelay_MF GnPortUI_MF

Choosing a Derivation Point

`GnSNMPAppDerPt` includes the basic functionality needed for an application model type. `GnChassisDerPt`, `GnDevIODerPT`, and `GnRelayDerPt` are derived from `GnSNMPAppDerPt` and therefore inherit this functionality. Each also includes some specialized functionality geared towards managing ports and boards.

If your device does not need to manage ports and boards and you are only interested in expanding support, use `GnSNMPAppDerPt` to derive your application model. See Creating Application Model Types [Page 55] for complete instructions on how to do this.

If your device uses other MIBs to extend the functionality of MIB-II in order to manage ports and boards, you will need to use `GnChassisDerPt`, `GnDevIODerPt`, or `GnRelayDerPt`. Each of these derivation points uses model fragments that contain the attributes and intelligence needed to create port models. The following section, Board and Port Considerations

[Page 52], describes how to decide which derivation point is most appropriate for your port or board model type.

Board and Port Considerations

As a general rule, if the device you are attempting to model is a chassis (a device that it has multiple modules/boards that can be inserted into and removed) then you would use the `GnChassisDerPt` and `GnRelayDerPt` derivation points to create the application model type. These two derivation points will be used to model both the boards and ports found in the device. The intelligence of these derivation points will create both board and port models.

If the device you are modeling is not a chassis, then you would build your application model type from the `GnDevIODerPt` derivation point. The intelligence of this derivation point will create only port models (no boards) and associate the port models with the device model.

The structure and content of the relevant MIBs is important to consider. Chassis and data relay MIBs generally have a standard structure. A chassis MIB usually has a slot/board table. The index of the table represents which slot of the chassis the board is plugged into. A data relay MIB usually has two tables, a board table and a port table. The board table is indexed by which slot the board is plugged into; the port table typically has two indexes, a board index and a port number. Additionally, vendors have devised several variations to the standard structures described above.

Port–Oriented Devices

You will generally use the `GnDevIODerPt` to model port oriented non-chassis devices. You will find that most MIBs for these port-oriented devices conform to the structural requirements necessary to use `GnDevIODerPt`. The MIB must contain a port table, with at least one index, the port number. The intelligence associated with the `GnDevIODerPt` will execute a `read_next` (a `read_next` is analogous to the `get_next` SNMP call) on this attribute. For each successful read of the index attribute, a port model with the appropriate instance ID will be instantiated.

Chassis Devices

The structure of the MIBs associated with chassis devices is much more varied. The best way to examine the variations and how they affect the modeling of the device is to view what is required by the intelligence

associated with the GnChassisDerPt and the GnRelayDerPt derivation points.

GnChassisDerPt

The GnChassisDerPt is used to create an application model type that will become the chassis manager application. This application will be responsible for the creation and management of board models in the SpectroSERVER database. This chassis manager relies on three attributes (usually list attributes) for the information it needs: slot index, board type, and board label.

There can only be one chassis manager application instantiated or managed by the main device model. The chassis manager intelligence is expecting the MIB to have a slot or board table indexed by an integer value representing the slot into which a particular board is plugged. The intelligence will perform a read_next on this slot index attribute. For each successful read, the intelligence will create a model in the database to represent that board. Because the intelligence can only reference one index value, all boards in the chassis must have an entry in this single table of the chassis MIB.

In addition to finding which slot a board is plugged into, the manager intelligence will need to determine the board's type and label the board correctly (for displaying the board Icon in the Device view). This information is also determined by the board type and board label MIB attributes. It is not necessary that these attributes exist in the same table as the slot index attribute. All that is required is that the attributes exist in a table with the same indexing scheme as the table used to discover the boards.

It is possible that the MIB will have all the board information in non-list attributes rather than in a table. In this case, the information supplied within the MIB is for a single board and the slot index value is not really an index into a table, but simply an integer attribute that will return the slot that the board is located in. The chassis manager intelligence will test the slot index attribute. If it is a non-list attribute, a read will be used instead of a read_next to get the board number. If the slot index attribute is not a list attribute, the board type and board label attributes will not be list attributes.

GnRelayDerPt

The intelligence of GnRelayDerPt is used to model the ports on a chassis. This derivation point can be used in combination with GnChassisDerPt to

create one application model, or it can be used on its own to create an application model separate from the chassis manager.

The term *chassis support application* is used to describe an application built with `GnRelayDerPt`. This is because it provides support to the chassis manager application (such as modeling the ports for each board). Unlike the chassis manager application, you can have multiple chassis support applications instantiated under the main device model. This becomes important when you consider a chassis which has boards supporting different protocols.

Although all the boards may show up in the chassis' slot table, the data relay component of each board may be managed by a MIB corresponding to the appropriate protocol. It is necessary to have each of these protocol dependent MIBs modeled as separate application models (built from the `GnRelayDerPt` derivation point) so that the ports found on each of the boards can be discovered and modeled.

The typical structure of a data relay MIB has two tables, a board table and port table. The board table should not to be confused with the slot table used with the chassis manager, although in some cases they can be the same tables. The board table found in the data relay MIB will have an entry for each board supported by the MIB, typically indexed by the position of the board in the chassis. For example, if the data relay MIB in question is an Ethernet MIB, then any board that supports the Ethernet protocol (typically a repeater board) will have an entry in this MIB's board table. If a FDDI board is plugged into the chassis, the board will create an entry in the common slot table, but this new board will not show up in the Ethernet MIB's board table. Instead, it will show up in the board table of the FDDI MIB.

Along with the board table, the data relay MIB will have a port table. For each port supported by the MIB there will be an entry in this table. The tables often contain the status and statistical information for each port. The port table contains two indices, a board index and a port index. Because the port table contains a board index, the chassis support intelligence can associate the port models with the appropriate board models; the board index supplies the mapping of a port to a board.

`GnDataRelay_MF` is the model fragment within the `GnRelayDerPt` derivation point that contains the attributes and intelligence which are used to model each board's ports and associate those port models to the appropriate board model. The intelligence associated with the `GnDataRelay_MF` model fragment works with only one board table and one port table. In the majority of cases, this is not a problem because this is

the typical structure of a data relay MIB. If your data relay MIB contains sets of tables, for example a set of board and port tables for each of the major protocols, then you must separate these tables or groups of the MIB into separate model types. To do this, use each model type as a base to the appropriate application built with the `GnRelayDerPt`.

There may be some cases where the data relay MIB does not have the typical structure of both a board table and port table, with the port table indexed to provide the physical mapping of ports to boards. This can be the case when the chassis device uses a MIB with a different indexing scheme for accessing the port information. An example of this would be the FDDI MIB which indexes the port table by the `SMTIndex` and the `PortIndex` (the `SMTIndex` has nothing to do with which board the FDDI port is physically located).

This situation can also be created if a vendor reuses a MIB from another device that it manufactures. The original device that the MIB was designed to manage was a port-oriented device (no boards, just ports). The vendor supplies the same functionality in a board that can be plugged into their chassis, and has decided to use the original MIB to manage the ports on that board. Its port table does not contain a board index, so there is no means of determining which board(s) has which port.

In this case, you should implement the `DataRelay_MF` model fragment functionality as you would with a port-oriented device. For further information on this, see Tutorial 2: Modeling Port-Oriented Devices [Page 157].

Creating Application Model Types

The following is a core set of tasks that you must accomplish when creating an application model type:

- Import Required MIBs [Page 56]
- Derive the Application Model Type [Page 56]
- Setting the default_attr or the default_attr_list [Page 57]
- Set the Model_Name [Page 59]
- Map Model Fragments [Page 59]
- Set Model Type Flags [Page 60]

You use the Model Type Editor to accomplish each of these tasks. The following section is designed to help you understand why the above tasks

are necessary, while the tutorials in Appendix D discuss how to implement these tasks in the Model Type Editor. You can also refer to the *Model Type Editor User Guide (0659)* for further information.

Import Required MIBs

When you create an application model type you may find that the MIB model type already exists in SPECTRUM (as with the rfc1368Mib model type used in Tutorial 2: Modeling Port-Oriented Devices [Page 157]), or you might need to provide access to the new MIB. If you need to provide access to the new MIB, you can do it in one of two ways. Use the Model Type Editor to either import the MIB directly into the new application model type, or create a MIB model type. If the MIB will be derived into multiple model types, it may be advantageous to derive the MIB into a separate model type that can be used as a derivation point, thus allowing the attribute IDs to be maintained across the model types. Organizing this MIB model type under a new or existing vendor model type helps to keep the database organized, as shown in Figure 7 [Page 37].

To create a MIB model type, derive a new model type from `GnSNMPMibDerPt` (see Figure 7 [Page 37]). Import the compiled MIB (see Importing the Liebert UPS MIB [Page 154] for instructions), and enter the proper SMI (structure of management information) path. Note that if the wrong SMI Path is used MTE will not produce an error. However, when viewing imported attributes the OID Prefix value will be incorrect. If the MIB imports without producing an error, you have created a new model type successfully. This process is detailed in Tutorial 1: Modeling Non-Data Relay MIBs [Page 153].

Derive the Application Model Type

The following procedure describes how to derive an application model type. See Tutorial 1: Modeling Non-Data Relay MIBs [Page 153] for an example.

1. In the Model Type Editor, select **Find Model Type...** from the **File** menu.
2. Enter `GnSNMPAppDerPt` in the **Name or Handle** field.
3. Single-click the **Apply** button to bring up the `GnSNMPAppDerPt` model type.
4. Double-click the `GnSNMPAppDerPt` model type to select it as the current model type.

From this point in the SPECTRUM database, create a new model type that will be used to model the application.

5. Select **New Model Type** from the **File** menu and enter the name of the new application model.
6. Click **OK**.

Your new application model type should now be the current model type in the Model Type Editor. If you have created a MIB model type, you should now add it as a base model type to the new application model type.

7. Select **Add Base Model Type** from the **Edit** menu and add the MIB model type as a base model type for new application model type.
8. Click **Apply** and then **OK**.

The `new application` model type should now contain two base model types: the `GnSNMPAppDerPt` and the MIB model type.

Setting the default_attr or the default_attr_list

When a device model for a specific device is instantiated, SPECTRUM queries the Model Type catalog. Most application model types that are derived from `GnSNMPAppDerPt` are queried and the value of each of these model types' `default_attr` or `default_attr_list` attribute is retrieved. SPECTRUM then queries those attributes on the device's MIB. When a match is found between an attribute value retrieved from the application model type and the corresponding attribute value retrieved from the MIB, SPECTRUM instantiates a model of this model type.

You can use either the `default_attr_list` or `default_attr` to specify attribute IDs from attributes of a MIB model type. The `default_attr_list` attribute allows you to specify multiple attribute IDs and the `default_attr` attribute allows you to specify one attribute ID. Each attribute allows SPECTRUM to identify the application model type that represents the MIB's functionality.

SPECTRUM performs a query of the attributes whose attribute ID is contained in the `default_attr` or `default_attr_list`. If `default_attr_list` is used, SPECTRUM will go through the list of attribute IDs and use the first supported attribute ID found to instantiate that application model to represent the MIB's functionality.

The `default_attr_list` attribute can be helpful if you have a device that supports just one table in a MIB rather than the entire MIB, and another device that supports other objects in the same MIB, but not in the

particular table that the other device supports. In this scenario, using the default_attr_list attribute to specify multiple attribute IDs ensures that the application model type representing the MIB will be instantiated for both devices even though they do not support the same objects in the MIB.

In order for this functionality to work in the application models that you create, you must set the value of the default_attr or default_attr_list correctly.

> **Note:** Aprisma suggests that you use an attribute from the MIB model type that represents a mandatory, non-list, external MIB variable when choosing a value for the default_attr or the default_attr_list attribute. This is especially important when creating a chassis application.

To set the value for default_attr:

1. Find the MIB attributes for the application model type you are working with.
2. Use the attribute ID of this attribute to set the value of the default_attr attribute in the application model type. You will need to look specifically at the attributes of the model type that represents the MIB. You can find the attribute IDs of a model type's attributes in the Model Type Editor Attribute view.

To set the value for default_attr_list:

1. Find the MIB attributes for the application model type you are working with.
2. Use the attribute IDs of these attributes to set the value of the default_attr_list attribute in the application model type. You can find the attribute IDs of a model type's attributes in the Model Type Editor Attribute view.
3. Set the Model_Group attribute to the decimal value of the application model's model type handle.

   > **Note:** It is important to make sure that the value of Model_Group is set appropriately. If Model_Group is set to 0, SPECTRUM will only use the default_attr attribute to identify the application model type that represents the MIB's functionality.

You must set the `default_attr` or `default_attr_list` in all application model types. An example of setting the `default_attr_list` can be found in Tutorial 1: Modeling Non-Data Relay MIBs [Page 153].

Set the Model_Name

Set the application model type's `Model_Name` attribute to the appropriate value. By default, this value will be used as the model name for any application model of this type. To do this:

1. In the Model Type Editor, go to the application model type's **Attributes View**. Scroll through the **Attribute Names** list until you find the `Model_Name` attribute.
2. Click on the `Model_Name` attribute to select it and choose **Alter Value** from the **Edit** menu. The **Alter Value** dialog box appears.
3. In the **Alter Value** dialog box, enter the appropriate model name for your application model.
4. Click **OK**.
5. Select **Save All Changes** from the **File** menu.

Map Model Fragments

If your new application model type is derived from `GnChassisDerPt`, `GnDevIODerPt`, or `GnRelayDerPt`, you must work with the model fragments that correspond to these model types to ensure the correct operation of port and board management. For a model fragment to function properly, you must map MIB attribute values from the application model type to model fragment attribute values using the Model Type Editor. This gives the model fragment access to information from the MIB it uses to create and manage ports, boards, and interfaces.

For example, one of the attributes required for the `GnChassis_MF` model fragment used with the `GnChassisDerPt` derivation point is the `boardIndex_Attr`. This attribute allows the model fragment to discover which boards are present in a chassis. The `boardIndex_Attr` needs to be set equal to the index attribute value in the board/group table of the chassis or repeater MIB. The index attribute usually returns an integer value or a series of values that represents a board number.

Certain derivation points have associated model fragments. The attributes associated with that model fragment are available to any model type based on these derivation points. If you need the functionality of a model

fragment that is not included with one of your base model types, include that model fragment as a base model type.

Appendix H: Model Fragment Reference [Page 111] defines all model fragment attributes. This appendix shows the model fragment values that must be set. Tutorial 2: Modeling Port-Oriented Devices [Page 157] and Tutorial 3: Building a Hub Manager Application [Page 173] show how to set model fragment attribute values using the Model Type Editor.

Set Model Type Flags

When creating an application model type it is important to set the value of a few different flags to ensure that models of this model type behave properly within SPECTRUM. These flags are available in the Attribute view of the Model Type Editor. Each flag represents a Boolean value and can either be selected (set to `TRUE`) or deselected (set to `FALSE`).

In most cases, you should select (set to `TRUE`) the `Visible`, `Instantiable`, and `Derivable` flags.

- If the `Visible` flag is set to `TRUE`, the model type is visible to all Model Type Editor users. If the `Visible` flag is set to `FALSE`, the model type is only viewable to a user with the same developer ID as the one used to create the model type.
- If the `Instantiable` flag is set to `TRUE`, you can instantiate a model of this model type in SpectroGRAPH.
- If the `Derivable` flag is set to `TRUE`, this model type can be used as a base for other model types.

In most cases, you should deselect (set to `FALSE`) the `No Destroy`, `Unique`, and `Required` flags.

- If the `No Destroy` is set to `TRUE`, users are not able to destroy a model of this type in SpectroGRAPH.
- If the `Unique` flag is set to `TRUE`, only one model of this model type can be instantiated in SpectroGRAPH.
- If the `Required` flag is set to `TRUE`, then a model of this model type must always exist in the SpectroSERVER database.

Saving Your Changes

Once you have finished deriving and configuring your new application model type in the Model Type Editor, you should save all changes in the

Attributes view. Then save the changes to the permanent catalog before exiting the Model Type Editor.

Modeling Ports and Boards

When you create application model types from `GnChassisDerPt`, `GnDevIODerPt`, and `GnRelayDerPt`, these applications automatically create the necessary port and board models needed to represent your device. SPECTRUM generally uses two model types to model these boards and ports, `GnModule` and `GnPort`. It is possible to derive new model types from them for customization purposes.

Modeling Boards with GnModule

Typically a board is modeled for one reason, to be a container for the port models that are physically located on that board. The board is displayed in the Device view, where the user can access GIB views to get board status and statistical information. In GnSNMPDev's chassis support, the `GnModule` model type is used to model many different types of boards.

GnModule Attributes

The `GnModule` model type can model multiple board types. `GnModule` has two attributes that help to define what type of board is being represented by a particular model, `gnType` and `gnName`.

The `gnType` attribute provides the board type as read from the chassis slot table. When each `GnModule` model type is instantiated, the chassis manager intelligence fills in the `gnType` attribute.

The `gnName` attribute is a text string that SPECTRUM displays as the label on the board model in the Device and DevTop views. This attribute is filled in by the chassis manager with information found in the chassis slot table when the board is first created

GnModule Icons

You are not restricted to using IIB icons provided with `GnModule` to display board models in various views. See Customizing Views [Page 92] and modulePibPrefix (1,2) [Page 121] for more information.

Deriving from GnModule

If you want to display board models in a view that does not support the method described in Customizing Views [Page 92] and modulePibPrefix (1,

2) [Page 121] you will need to derive a new model type instead of using `GnModule`.

All new board model types must be derived from the `GnModule` model type.

Modeling Ports with GnPort

Port models are very similar to board models, GnSNMPDev provides one port model type that should be sufficient for most modeling needs. The `GnPort` model type is the default model used to model ports using the GnSNMPDev's chassis support.

GnPort Icons

Like the `GnModule` model type, `GnPort` models can be displayed with icons other then the IIB files supplied with `GnPort`. They also can be used with Device and DevTop view. This provides the ability to display different information in the port icons (not just a status and counter provided in the GnPort icon), as well as allowing access to different GIBs from the port model icons.

Creating New Port Model Types

As with the board models, the only reasons to create new port model types is if your ports will be displayed in views which do not support the methodology outlined in Customizing Views [Page 92] and modulePibPrefix (1,2) [Page 121]. However, it is not necessary to derive your new port models from the `GnPort` model type.

Port and Board Model Information

This following information is not necessary for modeling your ports and boards, but is presented to enhance your understanding of how the information for each board and port is read and displayed in the icons and GIBs associated with each board or port.

All external attributes associated with the boards and ports are read through the application model(s) used to support the board and port models. This is because the application models contain the MIB model types and thus the external attributes which are associated with the boards and ports.

Any GIBs that show board and port information must be placed in the CsGib directory of the application model type, rather than in the CsGib directory of the port or board model type. This is because each icon is created in the view using the application model handle and not the model

handle of the board or port. The application's model handle is used because the application contains the external attributes to be read.

In addition, the actions in the IIB file that are used to create the menu refer to the application model type name in the CsGib directory rather than the board or port model type name.

For example, the SPECTRUM rfc1368App (the application model used with GnSNMPDev to model any chassis managed by the IETF SNMP- Repeater MIB), models a chassis using the `GnModule` and `GnPort` models. Each board and port modeled by the rfc1368App and displayed in the Device view has two GIBs accessible from the icon; a configuration GIB and a performance GIB. There is no CsGib directory for `GnModule` or `GnPort`. The GIBs used to show the board and port information modeled by the rfc1368App are found in a sub-directory for the application, i.e. `/SG-Support/CsGib/rfc1368App`.

Creating SpectroGRAPH Support Files for a New Application Model Type

The `mmbuild` tool lets you to create SpectroGRAPH support files for a model type. The `mmbuild` tool incorporates the new model type into the database, and then links it to all the SpectroGRAPH support files that it requires in order to be represented though the SpectroGRAPH. It also links the new model type to the files that allow the icons representing the new model type to appear in the other SPECTRUM views. For more information, refer to Appendix C: Creating SpectroGRAPH Support Files for New Model Types [Page 64].

Distribution

The SPECTRUM Extension Integration (SEI) toolkit allows you to create a virtual CD (VCD) that allows you to distribute an install new model types on other SPECTRUM hosts. For more information, see Appendix G: Distribution [Page 109].

Appendix C: Creating SpectroGRAPH Support Files for New Model Types

Once you have created the device or application model types necessary for modeling the device, you must create SpectroGRAPH support files. The support files allow instantiated models of new model types to be correctly viewed in the SpectroGRAPH. This section outlines how to use the `mmbuild` script to create support files.

In This Section:

[Running mmbuild [Page 64]]

[Deleting Support Files [Page 66]]

Running mmbuild

The `mmbuild` script incorporates the new model type into the database, linking the model type to all the required SpectroGRAPH support files. It links the new model type to the Perspective Information Block (PIB) files that allow icons representing the new model type to appear in the other SPECTRUM views. The script also copies blank Generic Views into the newly developed application model directory. In addition, device model types will have a newly created directory structure for AlertMap and EventDisp files. A new AlertMap and EventDisp can be created for proprietary trap support. See the AlertMap and EventDisp files that support the standard IETF MIB traps in `<$SPECROOT>/SS/CsVendor/IETF` for a reference. For more information on AlertMap files, see [Appendix E: Alert, Event and Alarm Processing [Page 105]].

The `mmbuild` script should be run for each model type you create, including both device and application model types.

Prerequisites

In order to run `mmbuild` you will need the following information for each new model type you have created.

- Base Model Type

- Model Type Name as defined in the MTE
- Developer Name as assigned by Aprisma Management Technologies

The Base Model Type name is the key `mmbuild` uses to determine which SpectroGRAPH support files should be used.

Entering an unknown Base Model Type name can cause unpredictable results for the model's representation in SpectroGRAPH. You will want to use the following base model types:

- **GnSNMPDev**: Use this base model type for device model types.
- **GnHubApp**: Use this base model type for application model types.
- **GnPort**: Use this base model type for port model types (see Modeling Ports and Boards [Page 61]).
- **GnModule**: Use this base model type for module model types (see Modeling Ports and Boards [Page 61]).
- **Gen_IF_Port**: Use this base model type for interface model types (see Modeling Ports and Boards [Page 61]).

You must be the owner of the SPECTRUM directories before starting the build.

Syntax

The `mmbuild` tool uses the following syntax:

`mmbuild [-f ] [-i ] <baseModelType> <modelType> <modelTypeVendor>`

`-f`: This is an optional parameter that indicates that any existing support files for the model type should be overwritten.

`-i`: This parameter is used for creating IIb files.

`<baseModelType>`: This is the base model type name used to derive the model type in the Model Type Editor (see the above section for further information on base model types). This is a required parameter.

`<modelType>`: This is the name of the model type derived using the Model Type Editor. This is a required parameter.

`<modelTypeVendor>`: This is the vendor name that you would like to associate with this model type. This is a required parameter. If you have been assigned a developer ID and a developer name by Aprisma Management Technologies, use your developer name. For more information on obtaining a developer ID, refer to the *Integrator Guide (5068)*.

Procedure

To run `mmbuild`, follow these steps:

1. Exit SpectroGRAPH and shut down the SpectroSERVER.
2. Go to the command line and navigate to SPECTRUM's `/SG-Tools` directory.
3. Use the syntax outlined above to run the `mmbuild` script. The following example runs `mmbuild` for a new model type called `myModel`, which has been derived from GnSNMPDev. Since the `-f` parameter is not used, any existing support files will not be overwritten. Since the `-i` parameter is not used, IIB files will not be created.

   ```
   mmbuild GnSNMPDev myModel VendorABC
   ```

4. As `mmbuild` runs, a list of files being created is shown on the terminal screen.
5. If `mmbuild` is successful, a message appears indicating that `mmbuild` succeed.
6. A log file is created that lists all of the created files and their locations. The log file is written to the `/SG-Tools` directory and is named `<modeltype>.log`, where `modeltype` is the name you provided at the command line.

Deleting Support Files

You can also use `mmbuild` to delete SpectroGRAPH support files that you have created. This is useful if a mistake is made while running `mmbuild` to create support files. The syntax for this is:

`mmbuild -d <modeltype>`

`<modeltype>`: This is the name of the model type whose support files you want to delete.

Appendix D: Customizing Views and Device Models

The user interface support files necessary to represent the device graphically can be edited in order to customize information that is displayed about the device. The device icon symbol, menu selections, and device views can all be customized.

In This Section:

Customizing Device Icon Symbols in the SpectroGRAPH [Page 67]

Customizing Alarm Manager and Search Manager Device Icon Symbols [Page 80]

Customizing Menu Selections in the SpectroGRAPH [Page 83]

Customizing Menu Selections in the Alarm Manager and the Search Manager [Page 83]

Implementing Conditional Menu Selections [Page 88]

Customizing Views [Page 92]

Customizing Device Icon Symbols in the SpectroGRAPH

When you use the `mmbuild` script, generic support files are created to display models in the various SpectroGRAPH views and in the Alarm and Search Manager. No modification of these files is necessary; however the icon image that appears on a device model and the menu choices available from a device model can be customized.

SpectroGRAPH, Alarm Manager and Search Manager use a number of support files to display models and information about those models in various SPECTRUM views. These files can be broken down into the categories outlined below:

- **PIB files**

  These are Perspective Information Block Files. They control which images can represent models in the Location, Topology and Device

Views. These files contain the directory paths that point to the images that are used in each view.

- **IIB files**

  These are Icon Information Block Files. They contain information about the icons that are displayed on a model. This information includes characteristics such as size, position, sub icons, and background image.

- **GIB files**

  These are Graphical Information Block Files. They control the contents of all of the generic views.

- **Image files**

  These are image files that control the background images displayed in the SpectroGRAPH.

- **Alarm Manager and Search Manager support files**

  The `mttpl.map`, `mm.tpl`, `.fig`, and `.isv` files work together to specify the appearance and behavior of icons in the Alarm Manager and Search Manager applications.

How Images are Selected for GnSNMPDev Device Icon Symbol

SPECTRUM determines the functionality of a device based on the applications that the device supports. As each application of a GnSNMPDev device is modeled, it registers its functionality with the GnSNMPDev model. Based on this information and through the use of various support files, SPECTRUM selects an appropriate image for the icon symbol representing the device model.

There are eight possible icon symbols that can be used by the device model:

- Generic Device
- Bridge
- Router
- chassis
- PC
- Terminal Server

- Work Station
- Switch

Model types derived from GnSNMPDev will also behave in this way. The following sections show how to modify this process to determine the image chosen for the device model icon.

The image_index Attribute

The device icon symbol displayed on a GnSNMPDev device icon, (also known as the sticky label) changes dynamically depending on the primary data relay function of the device.

An application model is created for a GnSNMPDev device model when SPECTRUM detects that a device has support for a specific MIB or portions of a MIB. If the application represents a type of data relay functionality, data from the application model is used to determine the device model's `image_index` attribute value. SPECTRUM interprets the value of the `image_index` attribute and selects the appropriate image for the icon. Many devices are easily re-configured. With a simple reset of the device, the functionality provided by that device can change, and the icon symbol will change to reflect the updated functionality.

A device may have multiple data-relay functions such as bridging and routing, or bridging and repeating. The question then is which of these data relay functions should be used to determine the device image that appears on the icon?

How the Value of the image_index Attribute is Determined

The value of the `image_index` attribute can be influenced by certain attributes in the `RelayFuncMF` or `StickyLabelMF` model fragments.

Application models that contain the `RelayFuncMF` model fragment are associated with their parent device model through the `Has_Relay_Function` relation. The device model's `StickyLabelMF` model fragment detects this type of association being added or removed. When this occurs, SPECTRUM intelligence re-evaluates the remaining applications and changes the icon symbol accordingly if a different application is now determined to be the most significant.

Customizing the RelayFuncMF Model Fragment

Application models that include the `RelayFuncMF` model fragment can influence the image selection for the device icon symbol. There are three attributes whose values affect this process.

- **isDynamic** (Boolean)

  This attribute describes the volatility of the `isFunctional` attribute. A value of `TRUE` indicates that the `isFunctional` attribute may change, and a watch must be placed on the `isFunctional` attribute (see below). Typically for applications that will be associated with GnSNMPDev models, this value will be set to `FALSE` indicating that the value of the `isFunctional` attribute is static and need not be watched.

- **isFunctional** (Boolean)

  The value of this attribute (`TRUE` or `FALSE`) indicates whether the `Has_Relay_Function` association should be made between the application model and the device model. The value of this attribute can be set in the application model type or can be toggled by some other intelligence attached to the application model itself. For a typical GnSNMPDev application model, the default value is `TRUE`.

- **RelayFunction** (TextString-16 chars)

  This is the keyword describing the main data relay function of the application. When the application model is created, this value is matched against the keywords in the `StickyLabelMF::OptionList` attribute of the device model to determine whether the image associated with the function should appear on the device icon.

  Note: The value placed in this attribute must also exist in the `StickyLabelMF::OptionList` attribute of the device model

In most cases, the only the `RelayFunction` attribute is customized. Set this attribute equal to the value that represents the type of data-relay function that this application performs. If this application will be associated with GnSNMPDev models, you are limited to the following: Default, SNMP, PC, WorkStation, TServer, Repeater, Bridge, Router, and Switch. If this application will only be associated with models of your new device type, then you can use any string as long as it also appears in the `OptionList` attribute in your new device type (see OptionList [Page 71]).

Customizing the StickyLabelMF

The `StickyLabelMF` model fragment is responsible for determining the type of device icon symbol on the device model. The intelligence associated with this model fragment sets the value of the `image_index` attribute according to the most significant data-relay function represented by the

associated application models. The StickyLabelMF model fragment has three attributes that affect the image_index attribute.

- **Enable_IH_Sticky**

  (Boolean)

  This attribute can be used to disable the functionality of this model fragment (see Disabling Automatic Device Image Selection [Page 78]).

- **Image_Attr**

  (Attribute ID)

  This attribute contains an attribute id which references an attribute that contains an integer representing a particular data relay function. The GnSNMPDev's Image_Attr contains the attribute ID for the tmp_image_index attribute (0x3d0002). The value of tmp_image_index is written to the image_index attribute unless the user has already specified a device type when creating the model manually.

- **OptionList**

  (TextString -128 chars)

  This text string indicates both the precedence and image_index value of the data relay functions that may be represented by associated applications. The value of the attribute is an enumerated pair of values, with a keyword and value pair for each entry in the list. Here is the string used as the default value for the GnSNMPDev's OptionList:

  "Default,0,SNMP,1,PC,5,WorStation,7,TServer,6,Repeater,4, Bridge,2,Router,3,Switch,8"

  The keywords from the string (PC, TServer, Repeater, Bridge, etc.) represent the possible values that will be read from the associated application models. The numeric values paired with the keywords are the index values to be written into the attribute referenced through the Image_Attr (for GnSNMPDev - tmp_image_index.)

  The position of the keyword/index value pair in the OptionList designates the precedence of that data relay function from lowest to highest precedence. When there are multiple applications associated with the device model, SPECTRUM uses the precedence to determine

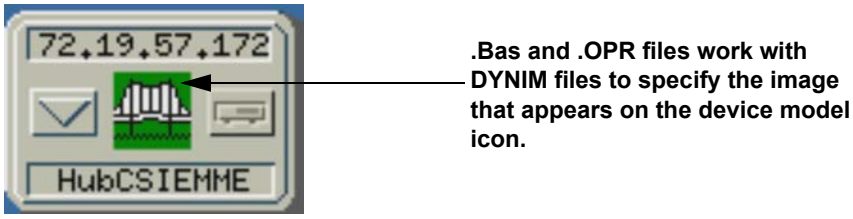
which of the data relay functions is most significant. The appropriate device icon symbol is then displayed on the device icon.

If you want to specify a different order of precedence for your new device model, you can just change the order of the pairs in the `OptionList`. Note that the first pair is the default choice. If no applications are associated with a device model, the default choice will determine the `image_index` value and, therefore, the device icon symbol displayed on the device icon.

Modifying Support Files

IIB (Icon Information Block) files are the basic building blocks of SPECTRUM views. There are many different kinds of IIB files that perform different functions. This section discusses three specific types of IIB files that work together to place the device icon symbol on the device model: `.Bas` files, `.OPR` files, and `.DYNIM` files.

Note: If you have created your own device model type and would like to customize the model icon, you must run `mmbuild` first to ensure that the proper support files are available. See Appendix C: Creating SpectroGRAPH Support Files for New Model Types [Page 64] for instructions.



.Bas and .OPR Files

Both `.Bas` and `.OPR` files contain a set of commands that control the appearance of model icons. The GnSNMPDev model type has both a `Large.Bas` and a `Small.Bas` file. The `Large.Bas` file is used for the Location, Device and Device Topology views. The `Small.Bas` is used for the Topology view. The `.OPR` file is used for an off-page reference in any one of these views.

The `.Bas` and `.OPR` files contain a command that is used to specify which image will be used for a device model icon. This command references a `.DYNIM` file that contains a list of image files that can be used as the icon

on the model. GnSNMPDev's Large.Bas file references the `Large.DYNIM` file and the `Small.Bas` file references the `Small.DYNIM` file.

Below is the `Small.Bas` file used to specify the appearance of GnSNMPDev models in the Topology view. It is located in SPECTRUM's `SG-Support/CsIib/GnSNMPDev` directory. The line that references the `.DYNIM` file is in bold. The syntax used in this line will be explained in the section on DYNIM Files [Page 73].

```
// Version: 1.1.1.3 Last Delta: 08/08/01 04:30:54
// Logfile: z:/cm/treebeard/sablime5_2/sdb/s604m/gnsmp/SG-
Support/CsIib/GnSNMPDev/s.Small.Bas
"NewRtr/grey4Sm.csi"
"NewRtr/greyb4Sm.csi"
ICON_PING
ICON_TELNET
{
      GnSNMPDev/DevDwn.BaMenu(63, 24, "Device", 0x3d0011,
10000, 1, "NWSimple", "Interface", "RTRDEVICE", 870, 800, 0,
"NWSimple", "Chassis", "GENDEVVIEW", 870, 800,0 )
      GnSNMPDev/Roll.LocMenu(7, 25, "DevTop", 0x3d0011,
10000, 1, "NWSimple", "Interface", "DEVTOP1", 780, 800, 0,
"NWSimple", "Chassis", "DEVTOP2", 780, 800, 0)
      Script.Act( 0,0,  GoPAView( "Performance", GENERIC ,
"CsPerform"))
      GnSNMPDev/SText.Ftx( 8, 47, 0x0023000e, "6x10", 245,
NWSimple( "Application", ZAPPVIEW, 870, 670 ) )
      GnSNMPDev/Small.DYNIM( 33, 19, 6, 0x003d0001,
GoPAView( "Performance", GENERIC, "CsPerform"))
SText.Txt( 8, 5, 0x0001006e, "6x10", 245, GoViewNW(
"Configuration", GENERIC , "CsConfig"))
      Script.Act( 0,0,  GoViewNW( "Model Information",
GENERIC , "CsStandard"))
      SelectPA.IPA( 0,0,GoAlarm ( "Primary Application"))
}
```

DYNIM Files

DYNIM files contain a list of image files that are used to provide the background colors and the device icon symbol on a device model. The supported image types are CsImage files and PNG (Portable Network Graphic) files. CsImage files (.csi) are proprietary Aprisma files used to create raster images. PNG files can be created or manipulated by many graphics software packages.

The DYNIM files for the GnSNMPDev model type can be found in SPECTRUM's SG-Support/CsIib/GnSNMPDev directory.

The .Bas or .OPR command that references the DYNIM file provides information on how the DYNIM file is set up. The following line from the GnSNMPDev Small.Bas file (referenced above) calls the Small.DYNIM file:

```
GnSNMPDev/Small.DYNIM( 33, 19, 6, 0x003d0001, GoPAView( "Performance",
GENERIC, "CsPerform"))
```

There are two arguments from this command that play a role in determining the image that will be selected: the offset value and the value of the attribute identified by the indicated attribute ID. The offset value is the number of background color images in the DYNIM file. In the example above, the offset is 6 and 0x003d0001 is the attribute ID of the image_index attribute. The offset value (6) is added to the default value of the attribute ID(0) to find the initial image to use (6+0=6).

Below is the Small.DYNIM file for the GnSNMPDev model type. Based on the sample command from the Small.Bas file above, to find the image file that will be used initially for the device icon symbol, start at 0 and count to 6.

```
0-> "GnSNMPDev/Small/sblnk_g.csi"
1-> "GnSNMPDev/Small/sblnk_b.csi"
2-> "GnSNMPDev/Small/sblnk_y.csi"
3-> "GnSNMPDev/Small/sblnk_o.csi"
4-> "GnSNMPDev/Small/sblnk_r.csi"
5-> "GnSNMPDev/Small/sblnk_w.csi"
6 use this image->"GnSNMPDev/Small/snmp_clp.csi"
"GnSNMPDev/Small/snmp_clp.csi"
"GnSNMPDev/Small/brdg_clp.csi"
"GnSNMPDev/Small/rtr_clp.csi"
"GnSNMPDev/Small/hub_clp.csi"
"GnSNMPDev/Small/pc_clp.csi"
"GnSNMPDev/Small/trmsrv_clp.csi"
"GnSNMPDev/Small/ws_clp.csi"
"GnSNMPDev/Small/atms_tp.csi"
```

When a GnSNMPDev model becomes active, a new value may be written to the image_index attribute (See How the Value of the image_index Attribute is Determined [Page 69] to find out about the factors that can change the image_index attribute value.) A new image_index value will

cause the image selected to change. For example, if the `image_index` value changed to `2`, the new image selected would be `brdg_clp.csi`, since offset value of `6` plus the `image_index` value of `2` equals `8`.

The image files referenced in the `Small.DYNIM` file are located in SPECTRUM's `SG-Support/CsImage/GnSNMPDev/Small` directory. The image files references in the `Large.DYNIM` file are located in SPECTRUM's `SG-Support/CsImage/GnSNMPDev/Large` directory. Note that each gives the path from the `CsImage` directory to the image file. The first group of image files are background colors and the rest of the image files will overlay the background image based on the value of the attribute ID (in the case of GnSNMPDev, the value of `image_index`).

Using a Custom Image

It is possible to use customized images in the image selection process. To do this, you must change the referenced image files in the DYNIM file(s) and add the customized images to the appropriate `CsImage` directories. You can use images that have been saved in the PNG file format.

If you are customizing a device model type that has been derived from GnSNMPDev, you must modify the DYNIM files and image selection appropriate to that model type. When `mmbuild` creates support files for new device model types, image files are located at `SG-Support/CsImage/<Model_Type_Name>` and IIB files are located at `SG-Support/CsIib/<Model_Type_Name>`, where <*Model_Type_Name*> is the name that you assigned the model type when running the `mmbuild` script. For more information on running `mmbuild`, see Appendix C: Creating SpectroGRAPH Support Files for New Model Types [Page 64].

For example, if you would like to change the device icon symbol that is displayed for all GnSNMPDev models that represent routers, you would need to make the following modifications:

1. Create the image you would like to use for the router models. This image must be in PNG format and should be the appropriate size to fit into the space provided on the model's device icon. For the purposes of this example, the PNG files are `MyLargeRouter.png` and `MySmallRouter.png`.
2. Copy the files containing this image into all of the appropriate `CsImage` folders. For the GnSNMPDev model type, `MySmallRouter.png` would be copied into `SG-Support/CsImage/GnSNMPDev/Small` and `MyLargeRouter.png` would be copied into `SG-Support/CsImage/GnSNMPDev/Large`.

3. Edit the DYNIM files appropriate to this model type so that they reference this new image. For this example the `Small.DYNIM` and `Large.DYNIM` files for the GnSNMPDev model type would be modified as follows:

   Small.DYNIM: Remove the reference to the Router image, (`"GnSNMPDev/Small/rtr_clp.csi"`), and replace it with a reference to the customized image (`"GnSNMPDev/Small/MySmallRouter.png"`).

```
// Version: 2.1.1.4 Last Delta: 06/30/94 18:46:59
// Logfile: z:/cm/treebeard/sablime5_2/sdb/gnsmp/SG-
Support/CsIib/GnSNMPDev/s.Small.DYNIM
// First 6 are the background colors and the rest are
the device clip masks
"GnSNMPDev/Small/sblnk_g.csi"
"GnSNMPDev/Small/sblnk_b.csi"
"GnSNMPDev/Small/sblnk_y.csi"
"GnSNMPDev/Small/sblnk_o.csi"
"GnSNMPDev/Small/sblnk_r.csi"
"GnSNMPDev/Small/sblnk_w.csi"
"GnSNMPDev/Small/snmp_clp.csi"
"GnSNMPDev/Small/snmp_clp.csi"
"GnSNMPDev/Small/brdg_clp.csi"
"GnSNMPDev/Small/MySmallRouter.png"
"GnSNMPDev/Small/hub_clp.csi"
"GnSNMPDev/Small/pc_clp.csi"
"GnSNMPDev/Small/trmsrv_clp.csi"
"GnSNMPDev/Small/ws_clp.csi"
"GnSNMPDev/Small/atms_tp.csi"
{
  Dummy.Ack (0, 0, Script ("Acknowledge", 0x00000000))
}
```

   Large.DYNIM: Remove the reference to the Router image, (`"GnSNMPDev/Large/rtr_clp.csi"`), and replace it with a reference to the customized image (`"GnSNMPDev/Large/MyLargeRouter.png"`).

```
// Version: 2.1.1.4 Last Delta: 06/30/94 18:46:55
// Logfile: z:/cm/treebeard/sablime5_2/sdb/gnsmp/SG-
Support/CsIib/GnSNMPDev/s.Large.DYNIM
// First 6 are the background colors and the rest are
the device clip masks
"GnSNMPDev/Large/blnk_g.csi"
"GnSNMPDev/Large/blnk_b.csi"
"GnSNMPDev/Large/blnk_y.csi"
```

```
"GnSNMPDev/Large/blnk_o.csi"
"GnSNMPDev/Large/blnk_r.csi"
"GnSNMPDev/Large/blnk_w.csi"
"GnSNMPDev/Large/snmp_clp.csi"
"GnSNMPDev/Large/snmp_clp.csi"
"GnSNMPDev/Large/brdg_clp.csi"
"GnSNMPDev/Large/MyLargeRouter.png"
"GnSNMPDev/Large/Hub_clp.csi"
"GnSNMPDev/Large/PC_clp.csi"
"GnSNMPDev/Large/TrmSrv_clp.csi"
"GnSNMPDev/Large/WS_clp.csi"
"GnSNMPDev/Large/atm_tp.csi"
{
  Dummy.Ack (0, 0, Script ("Acknowledge", 0x00000000))
}
```

You must restart the SpectroGRAPH for these changes to take effect.

Device Icons in Application Views

The device icon symbol that appears in the Application view is not manipulated via `.Bas`, `.OPR`, or DYNIM files. This model's appearance is controlled by the `.AIBase` file. The `.AIBase` file for the `GnSNMPDev` model type is called `GnSPApp.AIBase`. Below is the command from this file that determines which device icon symbol will appear on the device icon. The section highlighted in bold shows the `image_index` attribute, `0x003d0001`, and a series of integers and image names.

`mth.DynLbl(44,23,40,40,181,10000,1,252, 0x10000,` **`0x003d0001, 0,AISTxt,1,AISTxt, 2,AIBdgLbl, 3,AIRtrLbl, 4,AIHubLbl,5,AIPcLbl,6, AITrmSrv,7,AIWSLbl,8,AISwLbl`**`)`

SPECTRUM uses these image names to determine the appropriate device icon symbol image. The value of the `image_index` attribute determines the integer/image name pair that is selected. For example, if the value of `image_index` is `3`, the image paired with the value `3, AIRtrLbl`, is used for the icon. Each image corresponds with a specific device function, as shown below. Note that the order and appearance of these images is the same as the `.csi` image files listed in the GnSNMPDev DYNIM files (see DYNIM Files [Page 73]).

0 - AISTxt : Generic

1 - AISTxt: Generic

2 - AIBdgLbl: Bridge

3 - AIRtrLbl: Router

4 - AIHubLbl: Hub

5 - AIPCLbl: PC

6 - AITrmSrv: Terminal Server

7 - AIWSLbl: Work Station

8 - AISWLbl: Switch

It is not possible to add or substitute a new image into the list of images available in this command. If you have customized any aspect of the device icon symbol selection for the device icon in the Topology, Device, Device Topology, and Location views, it is suggested that you use the generic `AISTxt` image (shown below) for the device icon in the Application view.

To do this, change value of the appropriate image reference to `AISTxt`. In the following example, the image name for routers has been changed from `AIRtrLbl` to `AISTxt`. Device icons that would normally show the Router image will now show the `AISTxt` image.

```
mth.DynLbl(44,23,40,40,181,10000,1,252, 0x10000, 0x003d0001, 0,
AISTxt,1,AISTxt, 2,AIBdgLbl, 3,AISTxt, 4,AIHubLbl,5,AIPcLbl,6,
AITrmSrv,7,AIWSLbl,8,AISwLbl)
```

Note: If you use an attribute other than `image_index` in the `.Bas` and `.OPR` files to determine the image, the following syntax should be used in the `.AIBase` file:
`mth.AISTxt(29,20,60,37,118,10000,1,253,0x10000)`
This syntax will generate the icon shown above.

Disabling Automatic Device Image Selection

It is possible to disable the functionality that automatically selects an image for a device icon symbol based on the functionality of the applications supporting that device. You should only disable this

functionality if you are creating a management model for a device and you want the device icon to always have the same image no matter what type of functionality that device supports. The instructions below show you how to disable the automatic image selection functionality and make the default value of the pertinent attributes correspond with the image you want displayed.

1. In the Model Type Editor, select **Find Model Type** from the File menu.
2. Enter the model type you have created in the **Name** or **Handle** field.
3. Select **Examine Attributes** from the **File** menu.
4. Scroll through the attributes of this model type until you find the **Enable_IH_Sticky** attribute.
5. Double-click the **Values** field of **Enable_IH_Sticky**
6. Set the attribute value to `FALSE`.
7. The attribute that controls the device icon symbol is the **image_index** attribute of the GnSNMPDev attribute group. Scroll through the GnSNMPDev attribute group until you find the **image_index** attribute.
8. Double-click the **Values** field of **image_index**.
9. Set the default value of this attribute to the integer that best represents the data-relay function of your device type, as shown in the table below.

image_index value	**Device Type**
1	Generic Device
2	Bridge
3	Router
4	Hub
5	PC
6	Terminal Server
7	Workstation
8	Switch

10. The model name, as it appears on the GnSNMPDev icon, is controlled by the **DeviceType** attribute in the **CommonInfo** attribute group. This holds text string up to 32 characters long, but 14 characters is about the most that will fit legibly on the device model icon. Scroll through the **CommonInfo** attribute group until you find the **Device Type** attribute.
11. Double-click the **Values** field of Device Type.
12. Enter the model type name you want to appear on the icon.
13. Select **Save All Changes** from the **File** menu.
14. Exit the Model Type Editor.

Customizing Alarm Manager and Search Manager Device Icon Symbols

It is also possible to modify the device icon symbols that appear on device models in the Alarm Manager and Search Manager applications. If you have created a new device model type, you must run `mmbuild` (see Appendix C: Creating SpectroGRAPH Support Files for New Model Types [Page 64]) before making the following modifications.

Device Model Appearance

The device model icon in the Alarm Manager and the Search Manager is based on entries in three different support files, the `mttpl.map` file, the `mm.tpl file`, and a `.fig` file. The following sections explain how these files work together to display the device model.

Associating the Model Type with the Icon

The `mttpl.map` file maps model types to a template file. The `mttpl.map` file is located in `/SG-Support/CsResource/Templates`. If a line entry for the model type does not exist in this file, the default device icon will be displayed. The `mttpl.map` file has the following syntax:

```
tmpl=RoamAboutIcon mtype=Test2 mthandle=0xffff0002
```

`tmpl`: This is the name of the template that you want to use to display the model

`mtype`: This is the name of the model type

`mthandle`: This is the hexadecimal model type handle.

Any models created using your new device model type will be represented with the default device icon because they will not have an entry in this file. You can add an entry to this file that specifies the device icon template to use. This template can either be chosen from the core set of icons listed in the next section, or it can be a new template that you have derived from this core set.

Defining the Device Icon Symbol Template

In the `/SG-Support/CsResource/Templates` directory, you will find a number of `.tpl` files. These files define the templates that are used in the `mttpl.map` file. Templates are created using inheritance, which allows new templates to be derived from certain base templates. The `.tpl` files in this directory are used to derive the core set of templates. This core set of icon templates listed below.

BaseAppIcon: base application icon

BaseDevIcon: base Device icon (without Device Topology and Device view support).

UserIcon: user icon that appears on the Repair view

DeviceIcon: device icon (with Device Topology and Device view support)

BridgeIcon: bridge icon

GnSNMPIcon: generic SNMP icon

HubBdgIcon: hub bridge icon

HubIcon: hub icon

RouterIcon: router icon

TS_XypIcon: Xyplex icon

WorkStation_PC_Icon: workstation type icon

PhysicalAddress: physical address icon

FanoutIcon: fanout icon

FlagIcon: location view icons

OffPageReference: off-page reference icon

OrangeIcon: eight-side orange icon used for composite topology models

NetworkIcon: icon representing the network model type

TokenRingIcon: Token Ring icon

LanIcon: LAN icon

FDDIIcon: FDDI icon

LinkIcon: icon representing WA_link.

LandscapeIcon: landscape icons.

PortIcon: icon representing a port

VNMIcon: VNM icon

You can derive your own template from one of the above core templates. To do this you add an entry to the `mm.tpl` file. This file keeps track of all management module level icon templates. The basic syntax is as follows:

```
(MyIcon CsTemplate DeviceIcon

sticky.SymbolFile = snmp.fig

condition.Bind = " mname mtname sticky downarrow"

condition.Attrs = "0x1006e 0x10000 0x1000a 0x10013"

)
```

The first line maps the name of the derived template to the template that it is derived from. In this case, the new icon, `MyIcon`, is derived from `DeviceIcon`.

The next line places the graphic in the center of the device model. All of these graphics are `.fig` files and are located in SPECTRUM's `/SG-Support/CsResource/symbol` directory.

The third and fourth lines map attributes for display in different portions of the device model. For example, the model name, identified by the attribute ID `0x1006e` is placed across the top of the device model in the `mname` slot.

If you create an icon using this syntax, you can then associate the icon with your model type in the `mttpl.map` file, using `MyIcon` as the icon template, as in the following example:

```
tmpl=MyIcon mtype=Test2 mthandle=0xffff0002
```

If you use an existing template from the above list, you must insert the name of that template into the `mttpl.map` file, as follows:

```
tmpl=HubIcon mtype=Test2 mthandle=0xffff0002
```

Customizing Menu Selections in the SpectroGRAPH

It is possible to customize the menu picks available for a selected device model. SpectroGRAPH menu selections are determined by a series of IIB files. For information on how to edit these files to create new menu picks, see the Launching Third-Party Applications section of the *Integrator Guide (5068)*. For information on creating conditional menu selections based on a device model's attribute availability or application existence, see Implementing Conditional Menu Selections [Page 88].

Customizing Menu Selections in the Alarm Manager and the Search Manager

It is possible to customize the menu selections available in the Alarm Manager and Search Manager for a selected device model. Menu selections are determined in an `.isv` file specific to the model type of the icon displayed. To customize the menu selection for your device, you must create an `.isv` file and map it to your model type.

For information on creating conditional menu selections based on a device model's attribute availability, see Implementing Conditional Menu Selections [Page 88].

Creating an .isv File

All `.isv` files are located in SPECTRUM's `SG-Support/CsResource/actions` directory. Generally, their names represent the management module to which they pertain. Each line of an `.isv` file corresponds with a menu entry for the model type. Each line can designate a specific action. These are the supported actions:

- GoView
- GoPAView
- GoScript
- GoSubMenu
- GoHTML
- GoWeb
- GoWebEnv

You can specify various parameters for each of these actions. The general syntax for a line entry is:

`act="<actionname>" "<parameter>=<parametervalue>"`

Following is a list of parameters available to all actions.

- `menu`: The name to be used for the menu command.
- `wgt`: The area on the icon that will invoke the action on a double-click. Wgt names are specified in `.tpl` files.
- `submenu`: The name of the submenu where this menu will be created.
- `contextId`: If you want the menu button to be context-sensitive based on an attribute value, specify an attribute id as a value for this parameter.
- `contextValue`: This parameter works with the `contextId` parameter. If the value of the `contextId` parameter is equal to the value of the `contextValue`, the menu choice will be enabled. When multiple `contextValue` parameters are specified, the menu choice will be enabled if the `contextId` is equal to any of these values.
- `contextNotValue`: This parameter works with the `contextId` parameter. If the value of the `contextId` parameter is equal to the value of the `contextNotValue` parameter, then the menu will be inactive; otherwise it will be active. When multiple `contextNotValue` parameters are specified, the menu choice will be inactive if the `contextId` is equal to any of these values.

  If neither the `contextValue` nor the `contextNotValue` are specified, the menu will be active depending on whether the attribute is present in the model.

  The `contextValue` and `contextNotValue` cannot both be specified in the same line entry.
- `application`: If this parameter is present, then the menu choice will appear only if the application name matches the value of this parameter. To find an application name, see the `.prf` file the application saves in the root directory or SPECTRUM's `SG-Support/CsResource/preferences` directory. It is usually "`.<app_name>.prf`" file.

Following is an overview of each action's syntax along with information about relevant parameters.

- `GoSubmenu`: This action creates a submenu associated with a menu name. For example the following line would create a menu item called **GoSubMenu** which would show a submenu called **My menu**.

  ```
  act="GoSubMenu"  menu="My menu"
  ```

- `GoScript`: This action runs a script that is located in SPECTRUM's `SG-Support/CsScript` directory. You can use the `arg` parameter to pass arguments to the script. For example the following line would create a menu item called **Ping** which would run the script `CsPingScript`. Two arguments would be passed to this script, as follows:

  ```
  act="GoScript" menu="Ping" script="CsPingScript"
  arg="{scriptdir}" arg="{0x1027f}"
  ```

  In general you can pass the values listed below using the arg parameter. All argument values must be offset with curly braces or the value will be passed as a string rather than a variable.

  `{scriptdir}` = name of the script directory

  `{landscape}` = landscape handle in hex format

  `{modelhandle}` = model handle of the icon

  `{vnm}` = VNM machine name

  `{vnmsocket}` = VNM socket number

  `{ui}` = machine where the SpectroGRAPH is running

  `{uisocket}` = UI server socket number

  `{0x10274}` = any attribute id, which will be replaced with the attribute value of the model

  For further information on launching scripts from an `.isv` file, please see the *Integrator Guide (5068)* and the *Enterprise Alarm Manager User's Guide (2065)*.

- `GoView`: This action allows you to navigate to a different view. The following line creates a menu command called Configuration that sends the user to the Configuration view for that model type.

  ```
  act="GoView" menu="Configuration" view="GENERIC"
  gib="CsConfig"
  ```

  The following parameters can be used with the GoView action:

- `view`: Name of the SpectroGRAPH view type to bring up on the model.
- `gib`: GIB file name if the view name is "GENERIC". You do not need to use the .30 extension.
- `oid`: OID string.
- `comp`: Component type (device, port, appl, or landscape).
- `width`: Width of the window.
- `height`: Height of the window.
- `lscpRel`: Landscape relation handle to follow (only for landscape models).
- `oidAttr`: Attribute handle whose value represents the OID string.
- `gibAttr`: Attribute handle whose value represents the GIB file (only for device models).
- `goup`: If you need to read the relation first before going to the view, then this specifies the relation handle
- `goupDir`: Specifies the direction in which to read the model (Left or Right).
- `goupMType`: Model type that should be matched if there is more than one. You could give multiple comma-separated values if you want to match more than one model type.
- `mhAttr`: Use this parameter to specify an attribute that stores a model handle. This allows you to launch a view based on the model represented by the model handle stored in that attribute.

- `GoPAView`:This action allows you to navigate to an application view. The syntax for this action is the same as the syntax for `GoView` except that each of the parameters specify values from the primary application model of the device.
- `GoHTML`:This action allows you to navigate to an HTML file. The following parameters can be used:
  - `file`:The file name including relative path information from the SPECTRUM root directory.
  - `url`:The URL to the HTML file. This parameter is only used if the file parameter is not specified.

- `GoWeb`: This action opens a window from the SPECTRUM web applications. After the menu name field, you add a series of tag and value parameters that work in pairs. The tag parameter specifies the variable you will be working with, and the value parameter specifies the value for that variable. These tag parameter pairs are concatenated to create the URL.

  Here is a list of values that can be used with the tag parameter:

  - `ATTR`: Specifies that the value field will be an attribute id.
  - `ENV`: Specifies an environmental variable. Environmental variables can be defined in custom installation scripts. See the *Extension Integration Developer's Guide (0623)* for more information.
  - `STRING`: Specifies a regular string.
  - `LANDSCAPE`: Specifies a landscape handle.
  - `MODELHANDLE`: Specifies the model handle of the icon.

  A value parameter should be used with each tag parameter. The value assigned to the value field will depend on the tag field. For example:

  ```
  act="GoWeb" menu="Web Stuff" tag="ENV" value="CV_URL"
  tag="ATTR" value="0x1027f"  tag="STRING" value="&community="
  tag="ATTR" value="0x10009"
  ```

  This example will translate into the following URL:

  ```
  http://thud:1741/cview?device=134.141.52.5,community=private
  ```

  where `CV_URL` is set to `http://thud:1741/cview?device=` via a custom installation script.

- `GoWebEnv`: This action also starts a SPECTRUM web applications page using environmental variables; however, this action enables you to replace strings in the environmental variables using macros. The following example uses the `macro` parameter to replace three strings within the `RME_URL` environmental variable. The string values are replaced with the values of the specified attributes.

  ```
  act="GoWebEnv" menu="Resource" envVar="RME_URL"
         macro="@%host%@" tag="ATTR" value="0x1027f"
         macro="@%read%@" tag="ATTR" value=""0x10024"
         macro="@%write%@" tag="ATTR" value=""0x10024"
  ```

Where `RME_URL=http://thud:1781/rme?Device=@%host%@, ROComm=@%read%@,RWComm=@%write%@`

In addition to the `ATTR` value for `tag`, the following values can be used.

- `ENV`: Specifies an environmental variable. Environmental variables can be defined in custom installation scripts. See the *Extension Integration Developer's Guide (0623)* for more information.
- `STRING`: Specifies a regular string.
- `LANDSCAPE`: Specifies a landscape handle.
- `MODELHANDLE`: Specifies the model handle of the icon.

Mapping your Model Type to the .isv File

Once you have created an `.isv` file to specify menu commands, link this `.isv` file to your model type with the `isv.map` file. This file is located in SPECTRUM's `SG-Support/CsResource/actions` directory. The `isv.map` file contains a listing of `.isv` files referenced by model type handle and model type name. By default, your new model type is not listed there, and you must add it using the following syntax:

```
file=Myisv.isv mthandle=0xffff0002 mtype=Test2
```

`file`: This is the name of the .isv file you have created.

`mthandle`: This is the model type handle of your model type.

`mtype`: This is the model type name of your model type.

Implementing Conditional Menu Selections

The following discussion describes how to provide conditional menu picks from a device model icon based on various criteria. Note that the mechanism is different for SpectroGRAPH based icons and Alarm Manager and Search Manager based icons.

The model type's `.Bas`, `.AIBase`, and `.OPR` IIB files can be edited to provide conditional menu picks for the SpectroGRAPH based icons. See Customizing Device Icon Symbols in the SpectroGRAPH [Page 67] for more information on these files.

The model type's `.isv` file can be edited to provide conditional menu picks for the Alarm Manager and Search Manager based icons. See Customizing

Alarm Manager and Search Manager Device Icon Symbols [Page 80] for more information on .isv files.

The following methods include instructions for both types of icons.

Method 1: Optional Menu Selection

Use method to make a device model menu item either selectable or grayed-out depending on the current value of one of the model's attributes. The specified attribute can be either internal or external and can be a table value.

SpectroGRAPH Icons

The following syntax should be added to the appropriate IIB files for the specific device model type:

```
Script.ActMnu( 0, 0, <Attribute ID>, <Match Value>,
GoViewNW(<Menu Name>, GENERIC, <GIB File Name> ))
```

The value of the attribute specified by <*Attribute ID*> must match the <*Match Value*> in order for the menu item specified by <*Menu Name*> to appear as selectable. When <*Menu Name*> is selected, the GIB specified by <*GIB File Name*> will be shown. Note that the <*GIB File Name*> must include a path to the appropriate GIB file relative to the IIB file you are editing.

Example

```
Script.ActMnu( 0,0, 0x1007d, 1, GoViewNW( "ATM Modeling
Options",GENERIC, "../ATMClientApp/CsLinkConf") )
```

The "ATM Modeling Options" menu item will only be selectable if the model returns a value of "1" when the attribute 0x1007d is read. Otherwise, the menu item will be grayed-out.

Alarm Manager and Search Manager Icons

The following syntax should be added to the model type's .isv file:

```
act="GoView" menu=<Menu Name> view=<GIB File Name>
contextId=<Attribute ID> contextValue=<Match Value>
```

The value of the attribute specified by <*Attribute ID*> must match the <*Match Value*> in order for the menu item specified by <*Menu Name*> to appear as selectable. When <*Menu Name*> is selected, the GIB specified by <*GIB File Name*> will be shown. Note that the <*GIB File Name*> must include a path to the appropriate GIB file relative to the file you are editing.

Example

```
      act="GoView" menu="ATM Modeling Options" view="../ATMClientApp/
CsLinkConf" contextId=0x1007d contextValue=1
```

The "ATM Modeling Options" menu item will only be selectable if the model returns a value of "1" when the attribute `0x1007d` is read. Otherwise, the menu item will be grayed-out.

Method 2: Hidden Sub-Menu Selection

Use method to create a cascading menu on the device model icon. The menu is built dynamically based upon the successful reads of specified attributes. The attributes can be internal or external and can be table values. If the attribute value is successfully read, then the selection will be shown. Numerous sub-menu items may be defined in this manner.

SpectroGRAPH Icons

The following syntax should be added to the appropriate IIB files for the specific device model type:

```
      Script.AttrMu( 0, 0, <Menu Name>, <Timer Value>,<SHOWGRAY/
REMOVE>, <Attribute ID>, GoViewNW, <SubMenu Name>, GENERIC, <GIB File
Name>,<SHOWGRAY/REMOVE>, <Attribute ID>, GoVIewNW, <SubMenu Name>,
GENERIC, <GIB File Name>,…)
```

If the attribute specified by <*Attribute ID*> is successfully read, the menu specified by <*Menu Name*> or <*SubMenu Name*> will be selectable. If the read fails, then the selection is either grayed out (if `SHOWGRAY` is specified), or not visible (if `REMOVE` is specified). The <*SubMenu Name*> will bring up the GIB view specified by the corresponding <*GIB File Name*>. The <*Timer Value*> should be specified as 0.

Example

This example provides a base menu item, "ATM" and conditionally adds two sub-menu items cascading from the base if the attribute reads are successful:

```
      Script.AttrMu( 0, 0, "ATM", 0,REMOVE, 0x1007d, "GoViewNW", "ATM
Modeling Options", GENERIC, "../CsModelOpt",REMOVE, 0x1007f,
"GoViewNW", "ATM Configuration Options", GENERIC, "../CsConfigOpt )
```

Alarm Manager and Search Manager Icons

The following syntax should be added to the model type's `.isv` file:

```
act="GoSubMenu" menu=<Menu Name>
```

act="GoView" menu=<*Submenu Name*> view=<*GIB View Name*> submenu=<*Menu Name*> contextId=<*Attribute ID*>

act="GoView" menu=<*Submenu Name*> view=<*GIB View Name*> submenu=<*Menu Name*> contextId=<*Attribute ID*>

If the attribute specified by <*Attribute ID*> is successfully read, the menu specified by <*Menu Name*> or <*SubMenu Name*> will be selectable. If the read fails, then the selection is either grayed out (if SHOWGRAY is specified), or not visible (if REMOVE is specified). The <*SubMenu Name*> will bring up the GIB view specified by the corresponding <*GIB File Name*>. The menu specified by menu=<*Submenu Name*> is a submenu of the menu specified by submenu=<*Menu Name*>.

Example

This example provides a base menu item, "ATM" and conditionally adds two sub-menu items cascading from the base if the attribute reads are successful:

```
act="GoSubMenu" menu="ATM"

act="GoView" menu="ATM Modeling Options" view="CsModelOpt"
submenu="ATM" contextId=0x1007d

act="GoView" menu="ATM Configuration Options" view="CsConfigOpt"
submenu="ATM" contextId=0x1007f
```

Method 3: Application Existence Menu Selection

Use this method to make a device model menu item selectable depending on the existence of an application model. The following sample IIB entries show how you can implement this conditional menu selection. In this example, if the device supports the RFC2667App, the Tunnel Interfaces and the Tunnel Configuration menu picks will be available. If the device supports the RFC2737App, the Physical /Logical Tables menu selection will be available. This example was taken from Large.Bas IIB files for the Nortel Contivity and Enterasys Matrix E1 management modules. These are cascading menu picks, which works well if you have more than one view that you want to make available.

Example

```
Menu.RelMenu( 0, 0, "VPN Status", RELATION, 0x1001f,"RIGHT",
"RFC2667App",STARTACTION, GoRelMtNW,"Tunnel Interfaces","GENERIC",
"TunnelIF", 0x0001001f, "RIGHT", 0xc4005b,STARTACTION, GoRelMtNW,
"Tunnel Configuration","GENERIC", "TunnelCnfg", 0x0001001f, "RIGHT",
0xc4005b)
```

```
Menu.RelMenu( 0, 0, "Entity Tables", RELATION, 0x1001f,"RIGHT",
"RFC2737App",STARTACTION, GoRelMtNW,"Physical/Logical Tables",
"GENERIC", "EntityMibObjects", 0x0001001f, "RIGHT", 0xc40055)
```

Definitions

- `0x1001f` is the relation handle for the `MANAGES` relation.
- `RIGHT` is the side of the relation that the application resides on with respect to the device model.
- `RFC2667App` and `RFC2737App` are model type names used for the search criteria
- `0xc4005b` and `0xc40055` are the model type handles for the application that the view is to be launched on behalf of.

Customizing Views

There are several different types of views that are used to display information about a model. These views can be customized in various ways. The following sections outline the support files used to control the behavior and appearance of SPECTRUM views.

Controlling View Display with the CsViewControl File

SPECTRUM views are controlled by a file called `CsViewControl`. The `CsViewControl` file is located in SPECTRUM's `SG-Support/CsPib` directory and contains a line entry for each view that SPECTRUM displays. Each entry defines how to create the view, defines what model types are eligible to appear the view, and points to the appropriate PIB file. This PIB file defines the IIB files that are used to represent the models in that view. Each line has the following syntax:

```
View Type Name, Relation Name, Model Type Name, Model Name,
Perspective File Name, Menu Name, Model Type Handle, Creation
Field
```

For example:

```
LOCATION, Contains, World, World, CsLocation.pib, Location,
ox10040, Autocreate
```

`View Type Name`: This is the name of the view type that is registered with the SpectroGRAPH upon start up.

`Relation Name`: For a model type to appear in this view, it must take part in this relation.

`Model Type Name`: This is the view's model type name.

`Model Name`: This is the name of the model created from the view's model type.

`Perspective File Name`: This is the PIB file that defines how a particular model type will look when modeled with the specified view type.

`Menu Name`: This is the name that will appear on the New View Menu for this view type.

`Model Type Handle`: This is the handle for the model type. This is only used if the next field is set to AutoCreate.

`Creation Field`: If this field is set to `AUTOCREATE`, this view model type will be created at startup. If this field is set to `NOCREATE`, this view model type will not be created at startup.

The `CsControlView` file can be edited. The existing line entries should *not* be changed, but additional entries can be defined using the prescribed format.

Generic Information Block (GIB) Views

GIB views organize and display attribute information and statistics for a specific model type. The most common GIB views are the Configuration view, the Model Information view, the Performance view, and the Application view. You can customize each of these views by adding charts, graphs, and tables to display dynamic statistical information about the device. The statistical information is based on the attribute values available to the model type. You can also add buttons and static text to customize the navigation and look of the view.

Once you have created a GIB view to display standard or proprietary MIB objects, you can use conditional menu picks to make the view available to only the GnSNMPDev device models that support the MIB objects. See Implementing Conditional Menu Selections [Page 88] for complete instructions.

The GIB Editor is the tool used to edit existing GIB views or create new views. In-depth information and instruction on using the GIB Editor are available in the *GIB Editor User's Guide (0660)*.

Device View

Three types of Device views are available for management modules:

- **Interface**: The Interface Device view displays a representation of each interface in the device's interface table. Interface icons display the interface number, interface type, MAC address, interface status, and a gauge of interface activity.
- **Chassis:** The Chassis Device view is used only for devices that contain boards and ports. This view shows the boards and ports that are in the device's chassis. From here, statistics and information relative to a particular board or port can be viewed.
- **Physical:** The Physical Device view provides a detailed, graphically realistic representation of the device.

The Interface Device view is always available from the GnSNMPDev model type. A Chassis Device View is available if the model has associated board model(s). A Physical Device View is not available for the GnSNMPDev model. A developer can create custom images and IIBs to support a Physical Device View.

Adding Device Views

Each device view has a single entry in the `CsViewControl` file. This entry dictates which PIB file lists the model types that may appear in that Device view. There are six virtual Device views, each with its own PIB file. For a device model type to have multiple Device Views, a distinct virtual Device view must be referenced in the `CsViewControl` file for each separate Device View. The following table shows the virtual view names and corresponding PIB files. Each of these views may be used only once per model type.

Device View Name	Perspective File
GENDEVVIEW	CsGDView.pib
IRMDevV	CsIRMView.pib
PHYSICAL	CsPhysical.pib
RTRDEVICE	CsRtrDev.pib
TRDEVVIEW	CsTRDev.pib
ZDVVIEW	CsZDView.pib

Accessing the Device Views

The Device Views that are available from a device icon are determined by the IIB file for that icon. To access a particular Device View from a device icon, an entry for that view must be included in that icon's IIB file. For example, the GnSNMPDev icon as displayed in a Topology View is controlled by the `CsIib/GnSNMPDev/Small.Bas IIB` file. The line in this IIB file that determines which Device Views are available is (all on one line):

```
GnSNMPDev/DevDwn.BaMenu(63, 24, "Device", 0x3d0011, 10000, 1,
"NWSimple", "Interface", "RTRDEVICE", 780, 800,0, "NWSimple",
"Chassis", "GENDEVVIEW", 780, 800)
```

This sub-icon creates a cascading sub-menu, one item of which may be variably disabled based on the result of an action sent to the SpectroSERVER. In the above example, the action is `0x3d0011`. The GnSNMPDev model will respond to this action based on the model's participation in a `CONTAINS` relation. The `CONTAINS` relation with a device model on the left side usually indicates the presence of a chassis. The item that can be disabled is indicated by a `0` in the first position of the line. In the above example, the `Chassis` menu option will be disabled if the device is not a chassis. Use of the `.BaMenu` sub-icon for the Device view entry of an IIB is widely practiced and highly recommended.

In cases where all device views are always available the action can be `0x0`, and each menu item will have a `1` in the first position. This IIB entry for an EMME is (again, all on one line):

```
HubCSIEMME/DevDwn.BaMenu(6, 4, "Device", 0x0, 0, 1, "NWSimple",
"Chassis", "GENDEVVIEW", 740, 700, 1, "NWSimple", "Interface",
600 )
```

This EMME icon will always offer three device sub-views. Notice that the EMME uses a different virtual view for the interface-style device view than the GnSNMPDev model.

How the Device View Works

The device view provides the following functionality:

- Graphically provide information about a device's interfaces, modules, and ports.
- Allow users to navigate down the interface/port level to view or change the interface/port configuration
- Provide a single view from which the performance and status of all interfaces/ports of a device can be monitored.

To accomplish this, the Device View relies on the SpectroSERVER for accurate and up-to-date information. The view communicates with the SpectroSERVER through the SSAPI's action mechanism. This mechanism provides a way for SpectroGRAPH to make specific requests of the SpectroSERVER and receive specific responses. A response can be an indication of success or failure, or large amounts of statistical data or configuration information.

The Action Mechanism

The SpectroSERVER attaches special intelligence, called inference handlers, to certain model types. In addition to controlling how these model types behave in SPECTRUM, inference handlers also respond to actions sent from client processes. Each inference handler may register to receive one or more actions, which are known by their integer identifier, or handle.

The typical device view uses two actions through which all communications with the SpectroSERVER are achieved. The first action is known as a `GET_LIST` action. The response to this action is expected to be a list of either interfaces or modules and ports. The `GET_LIST` action also returns a integer stamp value which, by itself, is meaningless. However, if you compare it to another integer stamp value, it indicates if the device's configuration has changed. The second action is known as a `POLL_LIST` action. The SpectroSERVER responds to this action by returning the integer stamp value of the device.

The communication sequence, then, between the device view and the SpectroSERVER is as follows:

1. The Device View sends a `GET_LIST` action to SpectroSERVER.
2. The SpectroSERVER dispenses the action to the Inference Handler that has registered to receive it.
3. The inference handler in question responds to the message by creating a list of interfaces, or modules and ports, and returns this list along with the stamp value.
4. The Device View displays the interfaces or modules and ports. The stamp value is stored.
5. At a user-defined interval (usually 20 seconds), the Device View sends a `POLL_LIST` action to SpectroSERVER.
6. The SpectroSERVER dispenses the action to the Inference Handler that has registered to receive it.

7. The inference handler in question responds by returning the stamp value of the device.
8. If the stamp value is different than the one previously returned (which would indicate that the device configuration has changed), these steps are repeated starting at Step 1. If the status value has not changed, the steps are repeated starting at Step 5.

By keeping an integer stamp value that represents the device configuration, the SpectroSERVER minimizes the amount a data that must be sent to the device view for regular polls. Table 4 describes the actions available for GnSNMPDev model Device Views.

Table 4: Action Codes

Action Code	Description
0x3d0002	This is the default `GET_LIST` action. It returns a list of modules and ports with the PIB names embedded.
0x3d0003	This is the poll action to be used in conjunction with the `GET_LIST` actions in this table.
0x3d0004	This action is very similar to the default `GET_LIST` action. However, the response includes blank modules in the list to represent empty slots.
0x3d0006	This action returns a list of ports associated with the first module in the device's chassis. This action is used for the port-oriented Device View.
0x230004	This action returns a list of the interfaces in the MIB-II interface table.
0x230005	This action is used in conjunction with the `DEV_GET_LIST_ACTION`. It returns the status of the interfaces of the device.

The following is the `GnChasDev.DEV` file, which controls the Chassis Device View display.

```
20000  // View Polling time (1000 = 1 second)

22     // percentage top window is of view

"BANNER.230001" // sub view type

875    // view width

700    // view height
```

```
20     // offset X starting position
20     // offset Y starting position
197    // window background color
0x3d0003 // poll list action handle
0x3d0002  // get list action handle
horizontal  // View vertically or horizontally
0     // Number of elements per row or column
leftright  // Layout the boards right to left
pib_index  // Name of entry in CsVarTbl holding .pib indexing
value
{
}
```

The following is the `GnSNMPDev.DEV` file, which controls the Interface Device view.

```
20000    // View Polling time (1000 = 1 second)
32     // percentage top window is of view
"DVBANNER.230001" // sub view type
875   // view width
700    // view height
20     // offset X starting position
20     // offset Y starting position
197   // window background color
0x230005 // poll list action handle
0x230004  // get list action handle
vertical // View vertically or horizontally
4      // Number of elements per row or column
leftright  // Layout the boards right to left
TRUE   // Boolean to set up the set poll class
0x230008 // Inference handler poll_action
```

```
{

}
```

Device Topology View

The Device Topology view (DevTop view) provides port-level management functionality. In this view, users can specify port-level connections to other devices or LANs. There are two different styles of DevTop views. Most non-chassis devices will use the Interface Devtop view, DEVTOP1. Chassis type devices may use the Chassis Devtop view, DEVTOP2, to specify port level connections for non-MIB-II ports.

Unlike the Device View, the DevTop View is actually four views or "panels" that work together. Each of these views has its own PIB file.

- **Off Page Reference Panel:** This panel displays devices that are connected to this device, but whose connections are not resolved at the port level. These devices can be connected to a specific port.
- **Large Device Icon Panel:** This panel displays a large device icon that allows you to access the GIB views of the device.
- **Simplified Device View Panel:** This panel controls the ports displayed on the Connections panel. If you have a multi-board device, select a board from the image to display the corresponding ports in the connections panel.
- **Connections Panel:** This panel displays icons for models that are connected to specific ports. Network group icons that contain the models are also shown.

Accessing the DevTop View

GnSNMPDev's method of accessing its DevTop Views is very similar to how it accesses its Device Views. In GnSNMPDev's IIBs, a `.BaMenu` sub-icon is used to produce a cascading submenu with one menu item greyed out conditionally. If the represented device is not a chassis, the Chassis Device View menu item is disabled. A very similar icon, `.LocMenu`, is used to provide DevTop view menu selections. The `GnSNMPDev/Small.Bas` IIB contains the following line:

```
GnSNMPDev/Roll.LocMenu(7, 25, "DevTop", 0x3d0011, 10000, 1,
"NWSimple", "Interface", "DEVTOP1", 780, 800, 0, "NWSimple",
"Chassis", "DEVTOP2", 780, 800, 0)
```

The `.LocMenu` icon will send the `0x3d0011` action to the model, asking if there are board models in a `Contains` relationship. If the action returned is

FALSE, the cascading menu will have the Chassis item greyed out or disabled. Otherwise, both options will be enabled. If you want your management module to offer both views, you can replace the 0x3d0011 action with 0x0 and replace the zero (0) in front of the second NWSimple with a one (1).

Unlike the Device View, there is only one IIB file that determines how the DevTop view will look for both Interface and Chassis type views. The DEVTOP1 and DEVTOP2 are parameters to the DevTop view that indicate what type of view to display. This is different from the Device View's .BaMenu, which indicates a different PIB file for the selection of IIBs.

How the DevTop View Works

Unlike the Device View, which \relies on sending actions to the SpectroSERVER to get a list of boards and ports, the DevTop View reads two key attributes to get this information. The HU_Part_Mdl attribute contains a list of attached interfaces (with the HASPART relation). Interfaces in this list are represented by interface icons in the Connections Panel. The HUConnLst contains both a list of devices that are attached to each port and a list of devices that are connected to this device but whose port connection is yet unresolved. Items in the former list will be represented by device or LAN icons attached to Interface icons in the Connection Panel. Items in the latter list are represented by icons in the Off-Page Reference Panel.

> **Note:** When a user moves a device icon from the Unresolved Off-Page Reference Panel to the Connections Panel, specifying the interface to which it is connected, the item is removed from the list of unresolved connections, and placed in the list of items connected to the interface in question.

Although, the DevTop View does not depend on actions for a list of interfaces, it does use an action to determine when the attributes should be re-read and the screen refreshed. The action is identical to the POLL_LIST action used by the Device View. This action returns a stamp value which is compared to the stamp value from the previous poll to determine if there has been any change.

In a Chassis DevTop View, the Simplified Device View Panel depends on an action to get a list of boards from the SpectroSERVER. Although the same GET_LIST action that is used in the Device View can be used here, the Simplified Device View Panel icon is not interested in the ports. The GnSNMPDev's Simplified Device View Panel IIB uses the PARAMETERIZED

action (0x3d0005). The icon is already setup to pass in a parameter that requests just the board and slot information. This action will cause boards to appear in the little chassis icon in their proper slots. If this is not desired, you can use the 0x3d0002 action instead.

When a user selects a board in the Simplified Device View Panel (by double-clicking on it), the model handle from which the HU_Part_Models and HUConnList attributes are read is changed to that of the newly selected board. The Connections Panel then reads the new attributes, gets a new stamp value, and redraws itself. Remember that interface models are related to board models via the HASPART association just as they are related to device models. The Connections Panel always reads the HASPART relation whether the view is a chassis or interface view.

The Large Device Icon Panel merely displays the same device icon that is displayed in Location Views. Table 5 shows the four panels and their corresponding PIB files.

Table 5: PIB files for the DevTop View

DevTop View Panel	PIB File
Unresolved Off-Page Reference Panel	CsDevTopOP.pib
Simplified Device View Panel	CsSriPib.pib
Large Device Icon Panel	CsLocation.pib
Connections Panel	CsDevTopBk.pib

The IIB files used for the Simplified Device View Panel and the Connections Panel need some explanation. The following is an excerpt from the GnSNMPDev.Dev file, the IIB file for the GnSNMPDev DevTop View:

```
"Cable/up1.csi"
0x3d0002    // get list action handle
0x3d000c    // model type handle used to index into
CsSriPib.pib
// to draw hub pib_index
//  use pib index option
                {
DUMMY( 95, 730, 10, 730, 95, 905  )
```

```
DUMMY( 265, 730, 180, 730, 265, 905 )
DUMMY( 435, 730, 350, 730, 435, 905 )
DUMMY( 605, 730, 520, 730, 605, 905 )
DUMMY( 775, 730, 690, 730, 775, 905 )
DUMMY( 945, 730, 860, 730, 945, 905 )
DUMMY( 1115, 730, 1030, 730, 1115, 905 )
DUMMY( 1285, 730, 1200, 730, 1285, 905 )
. . .
DUMMY( 7745, 730, 7660, 730, 7745, 905 )
DUMMY( 7915, 730, 7830, 730, 7915, 905 )
DUMMY( 8085, 730, 8000, 730, 8085, 905 )
                }
```

The first entry, `Cable/up1.csi`, indicates the image that is to be used to draw connecting pipes in the Connections Panel. The third entry, `0x3d000c`, is used to index into the `CsSriPib.pib` only in the Chassis DevTop View. Otherwise, if the interface view is selected, the model type handle of GnSNMPDev is used. The fourth entry, `pib_index`, indicates that if an interface model's `pib_index` attribute is not empty, the contents of this attribute should be used to index into the `CsDevTopBk.pib` file, instead of the interface's model type. The DUMMY entries inside of the curly brackets only serve to place the interface models in the Connections Panel. There must be at least one DUMMY entry for each interface the may be displayed. If you are modeling a terminal server that could have 32, 64, or 96 ports, you should have at least 96 DUMMY entries in your DevTop IIB.

A sample of a Simplified Device View Panel IIB file, `SNMPHUB.GSISd` is shown below:

```
2x2/slot17.csi" //  Background Raster.
0x3d0002 //  SpectroSERVER action number.
20000 //  Icon Polling time (1000 = 1 second)
Left_to_right
            {
```

```
    DUMMY( 355,  52 )
    DUMMY( 334,  52 )
    DUMMY( 313,  52 )
    DUMMY( 292,  52 )
    DUMMY( 271,  52 )
    DUMMY( 250,  52 )
    DUMMY( 229,  52 )
    DUMMY( 208,  52 )
    DUMMY( 187,  52 )
    DUMMY( 166,  52 )
    DUMMY( 145,  52 )
    DUMMY( 124,  52 )
    DUMMY( 103,  52 )
    DUMMY( 82,  52 )
    DUMMY( 61,  52 )
    DUMMY( 40,  52 )
    DUMMY( 19,  52 )
}
```

The background raster image, `2x2/slot17.csi`, displays a small image of a 17-slot chassis. If your chassis has fewer slots, you may wish to create your own background raster. If your chassis has more slots, you must create your own background raster or you will not be able to display all the boards in the Simplified Device View Panel. Notice that like the `GnSNMPDev.Dev` IIB, this file has DUMMY entries. The number of DUMMY entries determines how many boards can be displayed in the Simplified Device View Panel.

Dealing with Double-Width Boards

It is possible to display double-wide board icons in the `SriPib` area. To do this, the model type used to represent the double-wide boards must be derived from `MultislotFrag`. You cannot simply create `GnModule` models

and manipulate the pib_Index attribute. The Num_Slots attribute in the MultislotFrag is needed to place the boards correctly.

To display your special multi-slot board models in the SriPib area, you need to create a special IIB file. A sample IIB file for a double-wide board is shown below:

```
"2x2/DblBlnk.csi"
"2x2/DblBlnkIn.csi"
0x3d0033
{
}
```

Use the 2x2/DblBlnk.csi and 2x2/DblBlnkIn.csi images if they suit your needs. The first image file in the Simplified Device View Panel is the image to display if the board is not selected. The second image is displayed (in the same spot) if the board is selected. 0x3d0033 is the attribute id of gnName. This text string is displayed in small letters on the icon.

Appendix E: Alert, Event and Alarm Processing

SPECTRUM is able to notify you about significant occurrences on your network through the use of alerts, events, and alarms. When creating a new model type, you must add support for alerts, events, and alarms.

In This Section

Alerts [Page 105]

Events [Page 106]

Alarms [Page 107]

Alerts

An alert is defined as an unsolicited message sent out by a managed node on a network. A more specific definition of an alert depends on the management protocol that is used to report the alert. In general, SPECTRUM uses SNMP as the management protocol to communicate with devices on a network. Alerts that are generated by an SNMP-compliant device are called traps. Traps are received by SPECTRUM and converted to events for further processing.

The AlertMap file that is associated with a management module contains the information on how SNMP traps should be converted into SPECTRUM events.

If you are enhancing trap support for the GnSNMPDev model type, you must create an AlertMap file located at:

`/SS/CsVendor/<nameofyourchoice>/`

Note: This will be a global AlertMap file.

If you add application model types to `GnSNMPDev` without creating a new device model, you must create an AlertMap file located at:

`/SS/CsVendor/<developername>/`

where *<developername>* = name of the vendor responsible for the application (i.e., your SPECTRUM developer ID name).

Note: This will be a global AlertMap file.

If you have created a device model type to represent your device, you will need to create an AlertMap file located at:

`/SS/CsVendor/<developername>/<modeltypename>/`

where *<developername>* = name of the vendor responsible for the device (i.e., your SPECTRUM developer ID name)

and *<modeltypename>* = the name of the model type that is being used to model the device (i.e., name of the device model).

For complete instructions on creating AlertMap files and for further information on global AlertMap files, see the *Event Configuration Files Guide (5070)*.

Events

An event is an object in SPECTRUM that indicates that something significant has occurred within SPECTRUM itself or within the managed environment. Events always occur in relation to a model. When a managed element on the network generates an alert, this alert is mapped to a SPECTRUM event in the appropriate AlertMap file. The event is then generated and takes on the event code specified in the AlertMap file.

EventDisp File

The management module's EventDisp file contains instructions on how the event should be processed. In this file you can specify whether an event should be logged, and whether it should play a role in creating or clearing an alarm. There are a number of different syntax options that can be used in the EventDisp file to specify when and if an alarm should be created.

If you are enhancing event processing for the GnSNMPDev model type, you must create an EventDisp file located at:

`/SS/CsVendor/<nameofyourchoice>/`

If you have added application model types to GnSNMPDev without creating a new device model, you must create an EventDisp file located at:

`/SS/CsVendor/<developername>/`

where *<developername>* = name of the vendor responsible for the application (i.e., your SPECTRUM developer ID name)

If you have created a device model type to represent your device, you will need to create an EventDisp file located at:

`/SS/CsVendor/<developername>/<modeltypename>/`

where <*developername*> = name of the vendor responsible for the device (i.e., your SPECTRUM developer ID name)

and <*modeltypename*> = the name of the model type that is being used to model the device (i.e., name of the device model)

For a definition of EventDisp files and their syntax, refer to the *Event Configuration Files Guide (5070)*.

Event Format Files

Once the event has been processed, information about the event can be displayed in the Event Log and the Alarm Manager. You can define Event Format files which determine the information to be displayed. Event Format files can contain textual information and can also contain variables that represent variable data (or VARBIND) sent with the SNMP trap.

For complete instructions on creating entries in an EventDisp file and creating EventFormat files, refer to the *Event Configuration Files Guide (5070)*.

Alarms

An alarm is an indication that a user-actionable abnormal condition exists in a model. A model usually detects an abnormal condition when an event occurs and the EventDisp file indicates that an alarm should be generated.

Once an alarm has been generated, information about the alarm is displayed in the Alarm Manager. This information is displayed via the Probable Cause file that is associated with that particular alarm. You will want to create a Probable Cause file for any alarm that you specify should be generated in the EventDisp file. The Probable Cause file is a text-only file that explains the possible reasons why the alarm occurred and suggests some solutions.

For complete instructions on creating Probable Cause files, refer to the *Event Configuration Files Guide (5070)*.

Appendix F: SpectroWATCH

The SpectroWATCH application can be used to generate events and alarms based on the results of a watch. You can create and distribute watches for your management modules.

All of the information that makes up a watch is stored as attributes in the model type specified in the watch. The only exception to this is the probable cause information created for an alarm resulting from the watch. This information is stored in a model type called `ProbCause`.

Because you have not created the `ProbCause` with your Developer ID, you do not have permissions to export and distribute it with the other components of your management module. This means that the probable cause information that relates to the watches you have created will not be distributed.

To solve this problem, you must derive a new model type from the `ProbCause` model type. The probable cause information for any watches created for any of your management modules will automatically be stored in this derived model type. Because you have created this derived model type, you can distribute it with your management module. For specific instructions on this process, refer to the *SpectroWATCH Operator's Reference (0919).*

Appendix G: Distribution

The SPECTRUM Extension Integration (SEI) toolkit allows you to create a virtual CD (VCD). This VCD enables you to distribute the new models types you have created to other SPECTRUM hosts. The SEI toolkit consists of a series of separate tools that are run from the command line, along with files that define the installable component. These tools and files are used to package the integration for distribution. The toolkit ensures that your integration is compatible with software from Aprisma and other Aprisma third-party developers. It allows customers to install an integration in their existing SPECTRUM environment with minimal impact from installation or integration compatibility issues.

In This Section:

Creating an Index File [Page 109]

Running mkmm [Page 109]

Running mkcd [Page 110]

Creating an Index File

The index file defines the components of the management module you have created, where they exist on your machine, and where these components will be installed on the customer's SPECTRUM machine. Index files include a reference to all files necessary for the operation and installation of the management module on the SPECTRUM host. Index files can be created manually or they can be created using the tool `mmship`. If you have not created a device model type and are trying to ship application model types that will support the GnSNMPDev management module, you will need to build your index file manually.

Running mkmm

Use the `mkmm` tool to build a package that contains all of the files referenced in the index file. This tool references the index file to create a VCD containing all of the necessary files to be installed on the customer's

SpectroSERVER. The `mkmm` tool is located in SPECTRUM's `/INSDK` directory, but it must be run from SPECTRUM's `/SG-Tools` directory where the index file is stored.

Running mkcd

The final step in creating an installable file is to use the `mkcd` tool which performs three main functions.

- It checks dependencies between purchasable parts and extensions modules on the VCD and reports any inconsistencies and errors.
- It adds a version number to the VCD, making the VCD installable.
- It locks the VCD against the addition of further files and prepares it for installation. The `mkcd` tool is located in SPECTRUM's `/INSDK` directory.

For in-depth information on working with the SPECTRUM Extension Integration toolkit, see the *Extension Integration Developer's Guide (0623)*.

Appendix H: Model Fragment Reference

This section contains a detailed description of all the model fragments used by the `GnChassisDerPt`, `GnDevIODerPt`, and `GnRelayDerPt` derivation points.

Many of the attributes in these model fragments have a suffix of "`_Attr`" (e.g. `boardIndex_Attr`, `boardType_Attr`, etc.). These attributes are of type `Attribute ID` which means that their values will be the handles of other attributes. You can think of them as pointers to attributes that have data of interest.

Each of the attributes described in this document are labeled with an IM, 1 or 2. These labels are used to indicate the purpose of each attribute, as follows:

- Implementation (IM)
- Level 1 modeling (1)
- Level 2 modeling (2).

Those attributes labeled for implementation are usually used by the device intelligence and are not meant to be changed by the developer.

The attributes marked for Level 1 and Level 2 modeling indicate attributes that may need to have their values set in order to accomplish the modeling of a particular device.

The GnChassis_MF Model Fragment

This model fragment is a base model type for GnSNMPDev's `GnChassisDerPt` derivation point. It is used, along with several other fragments, to model third party hub devices. The `GnChasssis_MF` model fragment contains all the attributes necessary for defining, creating and then managing the chassis (boards). It contains information on how to find boards within the chassis, how to get each boards type and name, what model type should be used to model each board, how often the chassis configuration should be checked, etc.

The `GnChassis_MF` model fragment also has the chassis manager inference handlers attached to it. This intelligence monitors the chassis device and watches for any change in the hub's configuration (new boards added,

pulled boards, etc.). The chassis manager ensures that the hub modeled in the database mirrors the actual device.

Below is a detailed explanation of each attribute in the `GnChassis_MF` model fragment.

aChassisManager (IM)

This attribute is used for implementation purposes. At run time, the value of this attribute is set to the model handle for the GnSNMPDev model (the device being modeled). It is used to facilitate communication between the chassis manager application and the chassis support applications. This communication is accomplished through the use of the `CsAction` mechanism. In order to send `CsActions`, the model handle of the intended recipient must be known. This attribute holds that model handle.

deviceMh_Attr (IM)

This attribute is used in initializing the chassis manager application when the model is first activated. The attribute contains the attribute ID for an attribute in this model type that contains the model handle for the main device model. It is used in filling in the `aChassisManager` attribute (see the previous section). By default, the attribute whose ID is used is the `physical_mh` attribute which is part of the `Gen_Create_MF` model fragment. When model creation occurs, the `physical_mh` attribute is set to the model handle of the device to which this application model is attached. If the value of this attribute is `0`, this indicates that the `GnChassis_MF` is being used as a base model type to a device model, as opposed to a base model type of an application model type.

resetOnChange (2)

This is a Boolean attribute that is used whenever a change is detected in the configuration of the physical hub. There are some hub devices that make a change to the ordering of port numbers assigned to all the boards in the hub if a board is added or removed from the chassis. Setting this attribute to `TRUE` in the model type will ensure that the chassis manager will check the configuration of every board in the hub whenever any change is detected in the hub configuration (i.e., it will ensure that the instance ID’s assigned to the ports on the remaining boards are correct).

configInterval (1,2)

This attribute contains the number of seconds between each check of the hub configuration. The default mechanism for managing the hub is an inference handler that runs every so often to determine if there is any

change in hub being modeled. The value of `configInterval` determines how often that configuration check is made. For example, the default setting of 600 means that a inference handler will run every 600 seconds to determine if the hub configuration has changed or not.

If you are using the `GnChassis_MF` model fragment to model a device which is not configurable (a device using a chassis MIB, but the boards are not removable modules) then a setting of `0` is appropriate for this attribute. This will ensure that SPECTRUM does not waste the time or bandwidth checking the configuration of a non-configurable device.

A force reconfiguration function is part of the chassis manager intelligence. The `CsAction` code of `0x3d002a` can be sent to the chassis manager (whichever model has the `GnChassis_MF` model fragment) to initiate a check of the device's configuration. This action is most useful from a GIB (Generic Information Block) associated with the chassis manager. The GIB can contain a button which will send the action off to the chassis intelligence (from GIB you must use the decimal equivalent of `0x3d002a` which is `3997738`).

boardIndex_Attr (1,2)

This attribute should be set with the attribute ID of an attribute that returns an integer value that represents a board number. Typically the referenced attribute is the index attribute in the board/group table of the chassis/repeater MIB. If the referenced attribute is an external, list attribute, the chassis intelligence will do a `readnext` on this attribute to find all the boards that are currently in the hub.

It is possible for the referenced attribute to be a non-aggregate attribute (the attribute does not have to be an index in a table). This non-table index attribute is simply a board number and can be either an external or internal attribute (an internal attribute can be used to imitate the presence of a board, to provide a SPECTRUM model on which to attach ports).

> **Note:** When each board is discovered in the chassis MIB, a model is created in the database to represent that board. The model type created is derived from the base model type `Module`, which contains the `Component` model fragment. The `Component` model fragment has an attribute called `Component_OID`, which is set with the value returned when reading the board index attribute from the device.

Each board model's instance ID can be set with either the value of the board index (a single term representing the board number) or with the

actual instance ID associated with the slot table entry. This is useful in cases where the slot table has multiple indexes.

boardTerm (1,2)

Some chassis MIBs actually have multiple indexes in the board or slot tables of the chassis MIB. This means that the slot number is not the only value used in indexing the slot table. The attribute `boardTerm` is used to specify which term of the instance ID represents the board number. By default this is set to `1`, because in most instances there is only one term in the instance ID - the board number. If the chassis MIB being modeled uses multiple indexes into the board/slot table, then this value may have to be changed depending on which index term represents the board number.

This attribute is not used in the discovery of boards in the slot table, but it is used to get the board number from existing boards. A board does not contain a board number attribute it only contains the instance ID of the board so, in the case of a multi-term instance ID, it is difficult to determine which term is the board number.

It is possible to select the how the instance ID (`Component_OID` attribute) for each board model is to be set using the `boardTerm` attribute. If the value of the `boardTerm` attribute is set to `0`, the value returned from the slot index attribute is used to set each board's instance ID. The instance ID for each board is a single value - the board number.

If the value of the `boardTerm` attribute is non-zero, then each board's instance ID will be set with the `suffix_oid` returned from reading the slot index attribute (this is the Object ID which represents the instance ID associated with each entry in the slot table).

If your device's slot/board table has multiple indexes, and you would like to access attributes from the table through GIBs or Icons associated with the board models, then you should set the `boardTerm` attribute to the appropriate value - the term representing the board number.

boardType_Attr (1,2)

This attribute should be set with the attribute ID of an attribute that returns a value that determines the type of board plugged into this slot of the chassis. The referenced attribute should exist in the same table as the `boardIndex_Attr`; the slot/board/group table of the MIB. The referenced attribute can be of any type supported by the SpectroSERVER database, but most likely the attribute will either be an integer or object id. This attribute is read by the chassis manager and used to determine the model type to instantiate when modeling this board in the database.

Note: The attribute referenced by `boardType_Attr` is typically found in the same table of the MIB as the board index attribute. However the attribute referenced here can be found in a different table of the MIB (or even in a different MIB), the only constraint is that both tables must be accessed using the same index (instance IDs). The type attribute is read from the device using the instance ID returned when the board index attribute is read.

If the attribute referenced by the `boardIndex_Attr` is not a aggregate attribute (from a table) then the attribute referenced by `boardType_Attr` cannot be an aggregate. This is because the type attribute is read with the same instance ID returned when reading the index attribute. If the index attribute is not truly an index, then there is no instance ID associated with the attribute.

boardType_Map (2)

The value of this attribute is used when the chassis manager needs to create board models. It used to determine which model type will be instantiated for each board discovered in the chassis device (see `boardType_Attr` above). As each board is found in the device, its type is read. That type is then used as a look-up value for this map to determine what model type to use when creating a board model in the database.

Note: It may not be necessary to map boards to model types. It may be sufficient to have all of your boards modeled as the default model type, `GnModule`.

This attribute is of the type `Text String`. The value of this string is read and interpreted by the chassis manager. The chassis manager will create the map from the contents of this attribute. There are two types of maps that can be specified by the user, `ENUM` and `RULE`.

A map of type `ENUM` is simply an enumeration of board type and model type handle, the map string would have the following format:

```
ENUM,default,bt,mth,bt,mth,bt,mth,...,bt,mth
```

boardType_Map ENUM Formats

- **ENUM**: String that identifies this map type as an enumeration type.
- **default:** This is the default model type handle, and is used when no entry is found for the particular board type being modeled.

- **bt:** (Board Type) A string representation of the value(s) that can be read from the board type attribute in the chassis MIB (`boardType_Attr`).
- **mth:** (model type handle) A string representation of the model type handle for the module or board model type to create when a board of type BT is found in the chassis.

  Note: The format used is one where each item in the string is separated by a comma and there are no intervening spaces between the comma and the next value. The board type (BT) values are the specified in decimal numbers, this includes integer board types as well as the individual terms of a Object ID board type.

The following map is the default map within the `GnChassis_MF` model fragment. It will create a `GnModule` model type (`0x3d000a`) for each board found in the hub. The following string is the default value found in the `boardType_Map` attribute:

```
ENUM,0x3d000a
```

The following are two example maps. In the first example board types are integers, the second example board types are object IDs.

```
ENUM,0x270015,38,0x270010,6,0x270056,18,0x270009,34,0x270023
```

```
ENUM,0xd0013,1.3.1.6.1.4.1.75.2.5.9,0xd0014,
1.3.1.6.1.4.1.75.2.5.18,0xd0038
```

In the first example, the default model type used in modeling a board is `0x270015`. If a board of type `38` is found in the chassis, the model type associated with the handle `0x270010` is created; if a board of type `6` is found, a model type of `0x270056` is created, etc.

The second type of board map that can be defined in the `boardType_Map` is a `RULE` map. This map is based on SPECTRUM's relations and hierarchical database structure. To use this rule, the developer should have a model type created for all board types that can be plugged into the hub being modeled. The user would also have to setup a relation in the database that can be used to find these boards, for example a rule such as `AcmeHubApp Contains AcmeBoards`. The contents of a RULE type map then instructs the chassis manager on how to find all the possible board model types in the database that can be used to model this particular type of hub.

The syntax of the string used to define the RULE map is the following:

```
rule, def_mth, relation, left_mth, type_attr, group_attr,
prec_attr
```

boardType_Map RULE Formats

- **rule:** This string specifies that the map to use is of the type `RULE`.
- **def_mth:** This is the model type handle of the port model type that is created when no match is found in the map for a particular port type.
- **relation:** This is the relation handle of the rule used in mapping out the port types. For example, the `Contains` relation is used, then the value used here would be `0x10001`.
- **left_mth:** This is the model type handle of the of the model type specified on the left side of the rule that you set up. For example, if an `ACME` hub is going to be modeled, and there is an `AcmeHubApp` model type that acts as the chassis manager, then you would use the model type handle assigned to `AcmeHubApp- 0x750001`.
- **type_attr:** This is the attribute id within a board or module model type that contains a value that represents the board type. The value read from the referenced attribute is compared with the value read from the chassis MIB (`boardType_Attr`). When each module model type is created, this attribute's value should be set to the value that will be returned when reading the chassis MIB.
- **group_attr:** This is the attribute id within a board or module model type that contains a value used in classifying a set of boards. This group attribute, along with the precedence attributes, constitute a mechanism to obsolete or replace old model types. This value should specify which attribute within the module model type is used to specify the module group, usually this will be the `MIM_Group` attribute (`0x119bf`) from the `Module` model type.
- **prec_attr:** This is the attribute id within a board or module model type that determines the precedence of boards sharing the same Group value. The most likely value for this argument is the `MIM_Precedence` attribute (`0x119c0`) from `Module` model type.

  Note: The `GROUP ATTR` and `PREC ATTR` arguments of the RULE specification are optional. These two parameters are needed only in the event that you have set your database up so that model types can be replaced using the meta-rule scheme.

The setting of boardType_Map attribute if the RULE mapping scheme were used, would appear something like the following:

```
RULE,0x3d000a,0x10001,0x750002,0x118be,0x119bf,0x119c0
```

In our example, the default model type specified is the GnModule (0x3d000a), the rule relation is the Contains relation (0x10001), the model type to appear in the left side of the rule is the AcmeHubApp (x0750002).

boardName_Attr (1,2) and nameParsingCmds (1,2)

These two attributes are used together for the labeling of the board models.

The boardName_Attr attribute should be set to the attribute id of the chassis/repeater MIB attribute that returns a Text or Octet string containing the name or label for each board model. The referenced attribute is typically found in the same MIB group as the board index and board type attributes. The name string will typically be the manufacturer's product name for this particular board.

> **Note:** If this attribute is set to 0, then boards are named using a different scheme (see boardName_Map).

This attribute works in conjunction the nameParsingCmds attribute. The value returned when reading the name attribute from the device often contains more then just the manufacturer's name for the board. It is typical for this attribute to provide a full description of the module; manufacturer's name, revision level, textual description of the board, etc. The nameParsingCmds is used by the chassis manager, to determine which word of the description string contains the actual board name. For example, a typical value return from a descriptor attribute may look like this:

```
Model 8390 rev1.02a Ethernet Network Management Module with 12
ports
```

If the board is a 8390, and that is how you would like the board labeled in the Device View, then you would set the boardName_Attr to point to the descriptor attribute from the board/slot table and you would need to set the nameParsingCmds attribute to isolate the word 8390. The nameParsingCmds is simply a sub-set of vi commands that can be used to edit the text to produce the resulting word.

For example, in the above text string, it is necessary to set the value of nameParsingCmds to dwf D. The commands are - delete word (dw) which

removes the word Model from the string. The command `f` is used to move the cursor to the space between `8390` and `rev1.02a`. Finally the `D` command will delete the text from the current position to the end of the buffer resulting simply in the string: `8390`. This is the text that will be used to label the board model.

Another example, if each module description string followed a format similar to this:

```
50001-MGT: Ethernet Management Module .....
```

Where the name of the board is the first word of every description string, only the name has the character `:` as a suffix. Set the `nameParsingCmds` to the value `f:D` which instructs the hub manager to parse the string by moving the cursor to the first colon character (`f:`) then delete the text to the end of the buffer (`D`).

> **Note:** In order to use this scheme for labeling boards, the format of the text must be consistent for all boards that can be plugged into the hub. The string of commands that you place into `nameParsingCmds` must result in the single word that can be used to label each board.

If you are not familiar with vi, the manual for the Korn Shell vi command line editor is a very good reference. You can include repeat counts for any command just like vi, for example: `2dw` (delete two words), or `2x` which can be used to delete two character:

```
Cursor Movement: lhwWbBeE0$fF Edit Commands: xX~Dd
```

If the referenced MIB attribute returns a string with just the board name (not a full description) then the `nameParsingCmds` attribute is left `<empty>`, thus no editing of the string read from the device is required.

boardName_Map (1,2)

This attribute provides an alternative means of naming or labeling board models (see `boardName_Attr` and `nameParsingCmds` for the preferred method). This attribute is used when the chassis MIB does not contain an attribute that provides the board name in the form of a text string. This attribute is a text string that provides a mechanism for mapping board types to board names. It is essentially an enumeration string used to map values read from the MIB for board types to text strings to use in labeling the board model for the Device view.

The value of this attribute is read and interpreted by the chassis manager. The text supplied as the attribute value must appear in the pre-described

format, a set of values separated by a comma with no intervening spaces. The format of the name map is the following:

```
def_name,bt,bn,bt,bn,bt,bn....bn
```

boardName_Map Formats

def_name: This is the text string that will be used to label any board model for which an entry can not be found within this map. The default value used in the `GnChassis_MF` is the string `GnModule`.

bt: This is a string representation of the value returned when the board/ module type is read from the chassis MIB. The board type strings are the key or index values for the map, the chassis manager will read the type from the chassis MIB, convert it to a string then use that value to compare against each BT specified in the map. A board type (BT) entry should be provided for any type of board that can be plugged into the hub being modeled.

bn: This string represents the text used to label the board model. The value specified in the map will usually reflect the manufacturer's product name used for this board. A board name (BN) should be supplied for each board type (BT) specified within the map.

The following is an example of a `boardName_Map` where the type value read from the MIB is an integer:

```
UnKnown,102,TR0-24,6,ENMOD-10,23,SM-TR,9,RC-18B,...188,
LAN-ENT6
```

In this example, all boards modeled that do not have an entry in the map will be labeled as `UnKnown`. If a board of type 102 is plugged into the hub, the model will appear in the Device View with the label `TR0-24`, a board type of 6 will be labeled `ENMOD-10`, etc.

Using this attribute and naming scheme is complex then using the `boardName_Attr`. The information to create the `boardName_Map` must be gathered for each possible board that can exist in the hub being modeled. One of the negative aspects of this board naming scheme is that when new board types are provided by the manufacturer, they will not automatically be included in the map. This attribute must be updated to reflect any new board types that can be plugged into this hub.

modulesType_Attr (IM)

This attribute is used for implementation purposes. The value of this attribute is set with the attribute ID of the board/module attribute that

contains the board's type. The referenced attribute returns a value that is the same as the value of the module type attribute from the chassis MIB.

This attribute is used each time the SpectroSERVER is started in order to determine if any changes in the hub's configuration have been made while the SpectroSERVER was down. The chassis manager uses this attribute to ensure that the board model created in the database is of the same type as the one that exists in the physical device. It allows the chassis manager to determine if the boards of a hub where somehow switched around when the SpectroSERVER was not running.

The default value for this attribute is the attribute handle of a `GnModule's` (the default board model type) `gnType` attribute. The `gnType` attribute is used to distinguish `GnModules` on the basis of the type of board the `GnModule` is being used to represent.

modulePibPrefix (1,2)

This attribute can be used to display the board and port models more precisely in the Device and DevTop views. To understand the significance of this attribute you need to understand the workings of the Device and DevTop views. To display the proper icons in these views, SpectroGRAPH uses the model type name (the model type being the boards and ports being displayed in the view) as an index into the PIB file associated with the view. The PIB file acts as a sort of look up table, it will instruct the SpectroGRAPH where to find the proper icon in the IIB directory to display a particular board or port based on the model's model type name.

With GnSNMPDev hub support, it is no longer necessary to provide a unique model type for every board or port that requires modeling. Unless there are special circumstances, all boards can be modeled using the `GnModule` model type and all ports can be modeled using the `GnPort` model type. There are circumstances where you need to retain the ability to display boards and ports with different icons (IIB files). To accomplish this, the Device and DevTop views provide an alternative means for specifying which icons to use in displaying boards and ports, rather than exclusively using the model type name.

The `modulePibPrefix` attribute is used in conjunction with the board name to produce a text string that can be used as this alternative model type name. The board name is the resulting string from either using the `boardName_Map` or `boardName_Attr` attributes (see the previous sections). The board name is the text string that is written into the `GnModule's gnName` attribute. It is the text that you will see each board labeled with when viewing the board in the Device view. The `modulePibPrefix` should

be set to a string that, as the name indicates, is a prefix for all board names.

For example, if you were modeling a hub from the manufacturer `acme`, then one possible setting for this attribute would be `acme_`. Continuing our example, if the boards found in an acme hub were labeled such as `3382`, `8837`, `9383`, `...` then the resulting alternative model type names would be `acme_3382`, `acme_8837` and `acme_9383`. These values would be used by the UI to index into the appropriate PIB file, which would reference unique IIB files for these boards, even though they were all modeled with the `GnModule` model type.

> **Note:** The string that results from combining `modulePibPrefix` and `gnName` should not exceed 14 characters. This is the current restriction on model type names, it should be adhered to for our alternative model type names.

The use of this attribute is optional and is only necessary if you are going to use the `GnModule` model type to model the boards, and that the default IIB file for a `GnModule` is not sufficient for displaying your boards models in the Device view. The IIB file for a `GnModule` provides for access to two GIBs; one to display a module's configuration (for example board status attributes) and one to display a module's performance (statistical information). In most cases, access to these two GIBs should be sufficient. If more GIB views are necessary, the `modulePibPrefix` should be considered an alternative means of accessing special IIB and GIB files for each board unless it is necessary to create a model type for each board.

There is also an attribute in the `GnDataRelay_MF` model fragment that provides the same functionality as the `modulePibPrefix` attribute discussed here. The attribute in the `GnDataRelay_MF` model fragment is the `altPibPrefix` and, as the name implies, is an alternate to the `modulePibPrefix` attribute. Using the `modulePibPrefix` attribute or `altPibPrefix` attribute depends on how you will eventually model your hub device. This is determined by the number and structure of the MIBs that will be used to build the application model type(s) that eventually will allow you to model the device.

If you have a single MIB to model the hub device, then you will use `modulePibPrefix` in the `GnChassis_MF` model fragment. If you have multiple MIBs to model the hub (a chassis/common MIB and separate repeater/concentrator MIB(s), and the GIBs that will be accessible off of the boards in the Device view will be used to show information from the chassis MIB, then you want to use the `modulePibPrefix`. This is because you want to associate the external files for the boards with the chassis

manager application (the application model built around GnChasssis_MF model fragment).

If you are going to be making a separate application model type for your repeater/concentrator MIB and the GIBs off the board icons of the Device View reference that repeater application (i.e., board status and statistics are accessed through the repeater application) then you want to use the altPibPrefix attribute of the GnDataRelay_MF model fragment to help create the alternative model type names. This means that modulePibPrefix is left <empty>.

Associating the name of board with the repeater application makes it easier to replace the board icons (IIB files) if the repeater MIB is modified or replaced in the future. This is accomplished by building a new repeater application to replace the old one (using the new MIB to build the new application model type). In this new repeater application model type, a slightly modified altPibPrefix is used so that the same board models can now be displayed with different IIB files.

For example, consider the fictitious acme hub. This hub has both a chassis MIB and a repeater MIB. You would need to build two application model types to model the acme hub. In one application model type, you would use the GnChassisDerPt and the acme Chassis MIB to build the chassis manager application. In this chassis manager application, the modulePibPrefix is left as <empty>. You would then use the GnRelayDerPt and the acme repeater MIB to create another application model type. In this application model type, the altPibPrefix attribute of the GnDataRelay_MF model fragment is set to acme1.0_. To complete the management module, you would have several IIB directories to build icon files (IIB files) for each of the boards that can show up in the acme hub (IIB entries - acme1.0_9938, acme_1.0_8872, ...).

Acme then decides to upgrade its firmware for this hub and they release a new MIB, the acme2.0_repeater_mib. This new MIB has additional tables for board and port information that will require the creation of new GIBs. All you need to do is to make a new repeater application model type, using the new acme Repeater MIB. In this new model type, set the altPibPrefix to the string acme2.0_. Then make a new set of IIB files for several of the board types that are effected by this new MIB (IIB entries - acme2.0_9938, acme_2.0_8872, ...). Now model the new acme hub with the 2.0 repeater MIB. In the Application view, a new acme repeater application has been created. In the Device view, there are new board models and new GIBs showing the new board table information. The old acme hub models that have not had the firmware modified, are still using

the original repeater application and IIB/GIB files. You never had to create a single new model type to model an acme board or port.

Selecting board icons on a per-application basis involves setting the `modulePibPrefix` and `altPibPrefix` attributes. Previously, the string contained in one of these two attributes was combined with the label of the board to produce an alternative index into the appropriate PIB file. The PIB index value is not produced on a board by board basis, but rather on an application or device basis. Developers want to specify an alternative icon (alternative to `GnModule's` icon) to be shared by all the boards.

If the string in the `modulePibPrefix` or `altPibPrefix` attributes contains a plus sign (+), then the concatenation of the PIB prefix string and the board name takes place. If no plus sign is contained within the `pibPrefix` string, then that string by itself is used for each board model's `pib_index` value.

For example, if the contents of `modulePibPrefix` is set with the value of `acme1.0Brds`, then all boards modeled for that hub will share the same board icon, the one referenced by `acme1.0Brds`. If on the other hand, the value of the `modulePibPrefix` attribute is set with a plus sign, `acme1.0_+`, each board modeled will have its own unique reference in the associated view's PIB file (`acme1.0_8803`, `acme1.0_8847`, ...).

The next three attributes described are attributes used for viewing the chassis in a management module's Device view. The three attributes; `slotCount_Attr`, `chassisType_Map`, and `blankPibIndex` can be setup so that the physical characteristics of the chassis being modeled can be determined dynamically. Previously, the Device views used by management modules could *not* show the full chassis view (e.g. no blank slots) because there was no mechanism to specify how many slots were in the chassis. Another disadvantage was that none of the management modules could support the Physical Device view without knowing how many slots are in the chassis. The addition of these three attributes, made this functionality available.

slotCount_Attr (1, 2)

This attribute is used to dynamically determine the number of slots in a hub chassis. How this attribute is used is dependent on the information available in the device's chassis MIB. The `slotCount_Attr` will be filled in with the attribute handle of another attribute, typically an external attribute associated with a chassis MIB. The number of slots in a hub is typically available from the device (the chassis MIB) from one of two

sources. The chassis MIB may contain an object which explicitly gives the number of slots in the chassis, for example:

```
chNumSlots OBJECT-TYPE
   SYNTAX INTEGER (0..64)
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
   "Number of slots in a chassis. For bounded,
   slot-less systems, the value of this object
   shall be zero(0)."
   ::= { chassis 3 }
```

It may be that your chassis MIB does not have an object such as `chNumSlots`. There are quite a few MIBs which have objects that specify the chassis's type from which the number of slots can be determined. Here is an example of such an object definition:

```
s3ChassisType OBJECT-TYPE
   SYNTAX S3ChassisType
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
   "The chassis type."
   ::= { s3000Chassis 1 }
```

If the chassis MIB that you are using to model the device contains an object to specify the number of slots in the chassis, then you reference the external slot count attribute by placing its handle into the value of `slotCount_Attr`. This is the simplest and most straight forward use of this attribute. Using our example object from above, after importing your chassis MIB into the database, you would take the attribute handle assigned to `chNumSlots` and use the handle to set the value of the attribute `slotCount_Attr`. Now the chassis manager intelligence knows what attribute to read to find the number of slots.

If, instead, your chassis MIB contains an object like `s3ChassisType`, which supplies the chassis type but not a slot count, then you would still set the value of the `slotCount_Attr` with the handle of the `s3ChassisType` attribute after the MIB has been imported. You also need to set up the `chassisType_Map` attribute (see the next section). The `map` attribute will be used to translate a type/value to the associated number of slots.

If the management module for your device will not show blank boards in the Device or DevTop view, then leave the value of the `slotCount_Attr` attribute set to `0`.

chassisType_Map (1,2)

This is a text string attribute. Setting this attribute is optional. It should only be used in the event that:

- You want blank boards in the Device view of your management module.
- Your chassis MIB has an object to describe the chassis type but *not* an object to determine the number of slots in the chassis.

This attribute provides the ability to map the chassis type value to its corresponding number of slots. The format of the `chassisType_Map` value is a list of pairs of chassis type and slot count. The following is the syntax for the value of the `chassisType_Map` attribute:

```
ENUM,default,type,slots,type,slots,type,slots
```

chassisType_Map ENUM Formats

ENUM: Used by the intelligence to determine format of rest of attribute value. Today, `ENUM` is the only allowed attribute format.

default: This is the default number of slots. This value is used if there is no explicit type entry found in the enumerated pairs that follow.

type: This is the textual representation of the value that will be read from the MIB object referenced through `slotCount_Attr`. Typically, the attribute/MIB object will be an integer, but it is not uncommon for MIBs to utilize an OID for this purpose.

slots: This field is the number of slots associated with a chassis of the preceding type specification. The following is an example of the contents of a `chassisType_Map` attribute. If the following object definition existed in your chassis MIB:

```
hubType OBJECT-TYPE

SYNTAX INTEGER {
```

```
ASE1000(1000), --.'ASE-1000'

ASE2000(3000), --.'ASE-2000'

ASE3000(5000), --.'ASE-3000'

ASE7000(11000) --.'ASE-7000'

}

ACCESS read-only

STATUS mandatory

DESCRIPTION

" The type of enclosure; "
```

To map the following type to the corresponding number of slots, the developer must set the value of the `chassisType_Map` to:

```
ENUM,0,1000,1,3000,6,5000,5,11000,12
```

This string indicates that a chassis type can have one of four values; `1000`, `3000, 5000` or `11000`. The default value is set to `0`, this is fairly safe in that if a new hub type cannot be identified, the Device view will simply not show Blank boards. In the example above, a hub of type `1000` has a single board and a hub type `3000` has `6`, a `5000` hub has `5`, and a `11000` hub has `12`.

Note: If the attribute referenced through `slotCount_Attr` points to a actual object which gives the number of slots (see `slotCount_Attr` description above) then the value of this attribute, `chassisType_Map` should be left <`empty`>.

blankPibIndex (1)

The `blankPibIndex` is the last of the three attributes used to show blank boards in a Device or DevTop view. This attribute is optional and is provided to minimize the amount of modeling required to accomplish this. You can use values other then the model type name when selecting icons to represent boards and ports in a new Device and DevTop view. This ability comes from having the model provide names or values to be used as indexes into the PIB files (see description of the `altPibPrefix` attribute in the previous section).

The value of this attribute can be left blank (<`empty`>) if the standard blank board icon is sufficient for displaying blank slots in the Chassis

Device view. This is the blank board icon associated with the `BlankMIM` modeltype (`modeltype handle = 0x000d0023`).

If your management module uses smaller or larger icons for boards (typically thinner or wider) then you can use the `blankPibIndex` attribute to specify the icon to use. Set this attribute with a value that is unique to your management module. You need to ensure that the value or name that you choose will not conflict with the name of an existing model type.

For example, if you were working on a management module for the latest acme hub chassis and the chassis was 20 slots wide, you may decide to make the board icons a bit thinner than the normal board icon. You may also decide to have the blank slots shown in the device's chassis view. When you set the attribute `blankPibIndex`, use the value `AcmeBlank`. In the `CsPib/CsGDView.pib` file, you then need the entry:

```
AcmeHub/AcmeBlank.DIBase,AcmeBlank
```

This entry in the `CsGDView.pib` file will ensure that the blank slots in Device view will be drawn using the icon specified in the file `AcmeHub/AcmeBlank.DIBase`. You can have the blank slots shown in the hub without having to create a modeltype with the name `AcmeBlank`.

The GnDeviceIO_MF Model Fragment

This model fragment is a base model type for GnSNMPDev's `GnDevIODerPt` derivation point. This derivation point provides a unique base model type for creating application model types used to model port/interface-oriented devices. Typically, these application models are used to model a device's ports that are not part of the MIB-II interface table. The `GnDeviceIO_MF` model fragment provides a mechanism for discovering, modeling and then associating these ports to the main device model.

The attributes within the `GnDeviceIO_MF` model fragment and the intelligence attached to this model fragment (inference handlers) provide flexibility when modeling any device with an SNMP agent. The model fragment provides mechanisms for the developer to instantiate different model types for each port or interface to be modeled. The model fragment also provides an attribute that allows you to select the relation to use in associating the port models back to the device model. This flexibility allows you to model many different devices.

This flexibility must be used with caution. Many of the views in SpectroGRAPH are dependent on the use of particular modeling schemes. For example, the DevTop view is dependent on port and/or interface models being associated with the `Haspart` relation. Fault isolation, auto

discovery, attribute value roll ups, etc. are also dependent on these modeling schemes. Use caution in selecting the model types and relations that will be used in modeling the device. Give careful consideration to what views you will be using to display and access the ports. Also give careful consideration in how the models you create will exist in the much larger Landscape, Topology or LAN models.

The `GnDeviceIO_MF model` fragment has the `GnDataRelay_MF` model fragment as a base model type. The `GnDataRelay_MF` model fragment is documented previously in the section on modeling repeaters as a part of GnSNMPDev's hub support, however some of the `GnDataRelay_MF` model fragment attributes are not used with `GnDeviceIO_MF`. Those attributes that are used will be discussed with regards to how they should be set for model ports or interfaces as opposed to boards.

The `GnDeviceIO_MF` model fragment is made up of the `GnDataRelay_MF` base model fragment. The attributes and intelligence for actually creating and associating port models exists in the `GnDataRelay_MF` model fragment. The attributes and intelligence associated with the `GnDeviceIO_MF` control if and when the device configuration is tested and, when necessary, request the `GnDataRelay_MF's` intelligence to model the ports and attach them to the main device model. `GnDeviceIO_MF` is the manager and uses `GnDataRelay_MF` in a supporting capacity. The `GnDeviceIO_MF` model fragment sends requests to `GnDataRelay_MF` using the `CsAction` objects. These actions request the `GnDataRelay_MF` intelligence to create port models, and to also check the current database model against the configuration of the physical device.

The `configurable` and `configCycle` attributes of the `GnDeviceIO_MF` are attributes that are used to control the initial as well as subsequent configuration management of the device. The `configurable` attribute is simply a Boolean attribute used to indicate if the device is configurable, i.e., can the number or types of ports change once the device has been modeled. If the device is configurable, the `configCycle` controls how often (in seconds) that the intelligence for the `GnDeviceIO_MF` model fragment will check the SpectroSERVER model against the actual physical device's current configuration. These two attributes control if and when configuration takes place, but the intelligence and attributes used in the actual configuration exist with the `GnDataRelay_MF` model fragment.

devicesMh_Attr (IM)

The contents of this attribute should be the attribute handle (ID) of an attribute which contains the model handle of the main device model. This value is then used when associating port or interface models created by

this model fragment to the main device model. By getting the device model's handle from an attribute, application model can be created and it can work independently of where the application model exists (as a `Manages` or `Provides` application).

It is also possible to use this model fragment as a base model type to the device model being constructed. If the value of `devicesMh_Attr` is set with the value of `0`, then the intelligence assumes that the `GnDeviceIO_MF` is part of the main device model type and not an application model type.

The default value for this attribute is the attribute ID of the `physical_mh` attribute within the `Gen_Create_MF` model fragment. This `physical_mh` attribute exists in all application model types and is set to the model handle of the device model when each application model is instantiated.

GnDeviceIO_MF Attributes

configurable (1,2)

This is a Boolean attribute which is read by the associated intelligence to determine if and when the configuration of the device must be tested. Some devices modeled with `GnDeviceIO_MF` will not be configurable, and will always have the same number and type of ports. Other devices are configurable, such as a device that has ports mounted on some type of removable module, or a device in which ports can be added or removed logically (though a local console).

If your device is not configurable, you should set this attribute to `FALSE`. When the attribute is set to `FALSE`, the intelligence associated with this model fragment will create and associate the port/interface models only once, upon creation of the device model. There is no periodic checking of the device's configuration.

If your device is configurable, then this attribute should be set to `TRUE`. A setting of `TRUE` ensures that the intelligence attached to this model fragment checks the device's configuration every time the SpectroSERVER is brought up, or anytime contact with the device is lost and then re-established. If the device can be configured dynamically, then you will need to set the `configCycle` attribute (see the next section) to ensure the intelligence monitors the device's configuration and makes the appropriate modeling changes when a configuration change is detected.

configCycle (1,2)

This is an integer timer attribute and should be used if the device you are modeling can be dynamically configured (changing the number, type or instances of ports while the device is up and running). The value of this

attribute is the time period, in seconds, between each test of the devices configuration.

For example, a setting of `600` would be used to have the configuration tested every `10` minutes. The model fragment intelligence will use the value of this attribute to register a timer to respond at the specified rate. Each time the timer expires, the intelligence will compare the devices configuration with the model in the SpectroSERVER database, if there are differences then the SpectroSERVER model is modified to represent the true configuration of the device.

The default setting for this attribute is `0`.

portGroupMth (1,2)

When it is necessary to associate the port models with the device through a board model instead of directly to the device model itself, this attribute contains the model type handle of that board model (which will be created by the `GnDevIO_MF` intelligence).

Because the DevTop view requires that the port models displayed in the view are associated to a model using the `Haspart` relation, it may be necessary to associate the port models to a board model to provide support for the DevTop view of the device. In the GnSNMPDev model type, this relation is used to associated the interface ports (from the MIB-II interface table) to the model. There are inference handlers associated with the GnSNMPDev model type that presume that any model associated with GnSNMPDev using the `Haspart` relation is in fact an interface port (such as `Gen_IF_Port`, `Serial_IF_Port` or `Data_Relay_IF` models).

The port models that you will be creating will not be interface ports and, therefore, will not be associated with device model via the `Haspart` relation. If you use the `Contains` relation to associate ports to the main device model, you will have a Device view but not a DevTop view for your device.

If you need to have a DevTop for the new port models, use the work around outlined below.

If the value of this attribute is left `0`, then all port models created for this device will be attached directly to the main device model using the relation handle specified in the `portAttachRel` attribute.

If the value of this attribute is not `0`, then it must contain a model type handle. The intelligence attached to the `GnDeviceIO_MF` model fragment will create a board model of this type, and the port models of this device would then be associated to this board model. The board model will in turn

be attached to the main device model (using the `Contains` relation). See `altPibPrefix` in the `GnDataRelay_MF` section to learn more about displaying these virtual boards in the DevTop and Device Views.

GnDataRelay_MF Attributes

The `GnDataRelay_MF` model fragment is a base model type for the `GnDeviceIO_MF` model fragment. There is a set of attributes within the `GnDataRelay_MF` model fragment that are not used when modeling devices with the `GnDataRelay_MF` model fragment, these attributes should retain their default settings. They are:

- managersMh
- useMapping
- portMap
- groupIndex_Attr
- groupTerm

The following `GnDataRelay_MF` model fragment attributes are used when modeling devices with applications built with the `GnDeviceIO_MF` model fragment:

- portIndex_Attr
- portType_Attr
- portAttachRel
- portType_Map
- portName_Map
- altPibPrefix

Refer to the section on the `GnDataRelay_MF` model fragment for a detailed discussion of each attribute (The GnDataRelay_MF Model Fragment [Page 135]).

portIndex_Attr (1,2)

The value of this attribute is the attribute ID (handle) of the index attribute from the port table of your device's MIB (more precisely - the MIB model type built from importing your MIB). The intelligence will execute a `read_next` on this attribute and for each index read, will create a port model.

The Component_OID for each port model created will be set with the suffix_oid returned by reading the index. The Component_OID is the instance ID used to reference this port in future reads by both the SpectroSERVER intelligence as well as the SpectroGRAPH icons.

portType_Attr (1,2)

This attribute should be used if the port table in the MIB contains an attribute to define the individual port types and you want to model each port type using a different port model type. The value of this attribute should be set with the handle (ID) of the external attribute in the port table that returns a value used to indicate the port type. The referenced attribute can be of any type supported by the SpectroSERVER. The intelligence reads the value and converts it to a text string, before using it in the portType_Map or portName_Map.

> **Note:** The type attribute is read using the same instance ID returned when the portIndex_Attr is read. The type attribute is read using a read as opposed to a read_next. This allows the type attribute to exist in a different table then the index attribute but the tables must be indexed in the same manner.

If the MIB you are modeling does not contain a port type attribute, this attribute should be set to 0.

portAttachRel (1,2)

This attribute contains the relation handle used in attaching port or interface models to the main device model. The default setting of this attribute is the HASPART relation (the value is 10004). This is the relation used to associate interface models to the main device model. If you have a device that contains ports which are not modeled as MIB-II interfaces, then you should set this attribute to the Contains relation handle (value of 10001).

> **Note:** Using the Contains relation allows you to view the port models and to access GIBs for each model, however, the Contains relation prohibits you from accessing these models in a DevTop view, thus preventing you from connecting devices to each of the ports.

If a DevTop view is required for the modeling of your device then it is necessary to set the attribute altPibPrefix (see below). This informs the intelligence to create an intermediary model to hold the ports (a GnModule

model). When you do this you must also set the `portAttachRel` attribute to the relation handle for `Haspart` (`10004`).

portType_Map (1,2)

This is a text string attribute used as a means of mapping port types read via the `portType_Attr` to model types. This map is simply an enumeration of port types to model types. Setting this attribute is required when you want to model different port or interface types with different model types.

The default setting of this attribute is the value "ENUM,0x3d000b". As explained in the section about the GnDataRelay_MF model fragment, this string is interpreted to mean that the GnPort model type should be used (whose model type handle is 0x3d000b) to model all ports. If you do not want to use the GnPort model type as the default model type, then simply replace the 0x3d000b string with the appropriate model handle of your port type. Or if you will be using the "portType_Attr" to read and map to multiple model types, then see the detailed instructions in the GnDataRelay_MF section.

> **Note:** If you are creating unique model types so that you can display them with special icons (IIB files,) refer to the description of the `portName_Map` attribute. This attribute provides a mechanism by which you can use an alternative means of indexing external files instead of using the model type name.

portName_Map (1,2)

This attribute gives you the ability to use something other than the model type name for indexing the external files of the SpectroGRAPH. You can use the GnPort model type to model all your ports. Then use this attribute to specify for each port, what name you want the port to be referenced by when displaying the port models in the Device and DevTop views.

For an in-depth discussion of this attribute, see the section on the `GnDataRelay_MF` model fragment (The GnDataRelay_MF Model Fragment [Page 135]).

altPibPrefix (1,2)

This attribute is only used if the `portGroupMth` attribute of the `GnDeviceIO_MF` model fragment is not blank, indicating that the port models are being associated with a board model and not directly to the device. In this case, the contents of this attribute will determine the `pibIndex` attribute of this board model, which specifies the IIB file that is

referenced in the Device and PIB files that are used to select an icon to represent the board in the DevTop view.

This is useful if you want to use the GnModule model type for the virtual board model, but do not want the GnModule icon to appear in the Device view. If you do not want the icon to appear, set this attribute to GnInvMd, which stands for Generic Invisible Module.

You can also set-up your own IIB file to display the module. In this case, you would need to use a unique name and enter it into the altPibPrefix attribute, provide an entry in the appropriate PIB file, and provide a directory and filename for the IIB icon definition.

Note: Previously, this attribute not only determined the pibIndex of the board model that the ports are attached to, but also determined whether or not a board model would be used for this purpose. If this attribute was <empty>, the port models would be associated directly to the device model. If the attribute was not <empty>, a model of type GnModule was created (and associated with the device model), and the ports were associated with the GnModule model. This was changed to allow the developer to specify which model type to instantiate for the board model.

The GnDataRelay_MF Model Fragment

The GnDataRelay_MF model fragment is a base model type for GnSNMPDev's GnRelayDerPt derivation point. It contains all the attributes necessary for defining and modeling the repeating component of a chassis or hub device. This model fragment is used in conjunction with other hub support model fragments to provide complete chassis/repeater modeling capabilities.

This model fragment is used to define the structure of a repeater MIB. The repeater MIB model type and the GnDataRelay_MF are often used as base model types for a new model type. The attributes within this model fragment are used to map out the repeater MIB for the intelligence attached to the new model type.

Attributes in this model fragment allow the chassis manager to discover the ports that exist on the boards plugged into the chassis, determine the type of ports, and also determine which board each port is located on. There is also an attribute within the GnDataRelay_MF model fragment that is used to determine what model type should be used to model each port

(if the repeater MIB describes the type of port, a different model type can be used to model each possible port type). The model fragment also contains information that provides a mapping of ports to boards. As the port models are created they must be associated with the correct board model to ensure that the Device view correctly reflects the device being modeled.

The functionality provided by `GnDataRelay_MF` is accessed through the use of `CsAction` calls. The `GnDataRelay_MF` intelligence provides services to the chassis manager by responding to `CsAction` calls initiated by the chassis manager. Such actions include getting a list of boards supported by the repeater MIB, requesting the service to populate a board with the appropriate number and type of ports, checking the configuration of the ports associated with a board, etc. The `GnChassis_MF` is used to model the chassis component of the hub, the `GnDataRelay_MF` is used to model the repeating component of the hub and together they provide the ability to model most third-party hub devices.

The following is a complete explanation of each of the attributes contained within the `GnDataRelay_MF` model fragment.

managersMh (IM)

This attribute is used for implementation purposes. It is used by an application model (an application that is acting as a hub service) to find the application model that is providing the chassis management functionality. When the model is activated, this attribute will be set with the model handle of the application model that contains the `GnChassis_MF` model fragment. This model is also associated with the device model using the `Manages` relation. While the SpectroSERVER is running, the value of this attribute is used by the application model to send messages (`CsActions`) to the chassis manager. Both the `GnDataRelay_MF` and `GnChassis_MF` model fragments can be combined in the same application model type. In this case `managersMh` attribute is the model handle for both the chassis manager and chassis support applications.

useMapping (2)

This Boolean attribute is used to determine how the inference handlers attached to this model fragment will map ports to boards. If the value of this attribute is `FALSE`, then the standard method of mapping ports to boards is used. The standard method is used if the repeater MIB contains two tables, a board/group table and a port table. The port table has two index values, one of which is the group or board number on which this port is located. There are attributes within the `GnDataRelay_MF` model fragment

that use this standard mapping of ports to boards when modeling the repeater. The default setting of this attribute is `FALSE` because the majority of the repeater MIBs are designed using this standard structure.

Some repeater MIBs do not have this standard format which provides this mapping of ports to boards. These MIBs often have only port tables and no way of knowing which ports are physically located on which boards. If this is the case, set this attribute to `TRUE`, and use the attribute `portMap` (discussed below) to provide the physical mapping of ports to boards.

portMap (2)

This text string attribute is used in conjunction with the `useMapping` attribute to provide a mechanism for mapping ports (found in the repeater MIB) to boards (found in the chassis MIB) when the repeater MIB does not provide this information. If the `useMapping` attribute is set to `TRUE`, then this attribute will contain the map for associating ports to boards. Use this attribute when the port table of the repeater MIB does not contain a board or group index.

This attribute's value can be set using the Model Type Editor or can be set dynamically by supplemental intelligence provided by the developer. Use the following format for setting the attribute value:

`BN,SP,EP,BN,SP,EP,BN,SP,EP,...BN,SP,EP`

Where:

BN: (Board Number) This is the number of the board for which ports are being mapped out. The board numbers can be in any order, and a board number can specified multiple times. Each board number (BN) has an associated starting port (SP) and ending port (EP).

SP: (Starting Port) This is the port number for the first port associated with the specified board. This value is matched against each of the port numbers returned from the device when a `readnext` is executed on the port table. If a match is made, it marks the beginning of a port group, all ports found in the device before a match is made on the associated ending port (EP) are assigned to the same board.

EP: (Ending Port) This is the port number that marks the end of a port group. As the device port table is scanned, reading a port index with this value indicates that all ports associated with the specified board have been found in the device.

The values used to define the starting and ending port numbers in the `portMap` text string are matched against the instance IDs (Object ID suffix)

returned when the port index attribute is read from the port table. If you are using this mapping scheme, it often means the that table has only one index, the port number.

However, it is possible for the table to contain multiple indexes. Often the additional index value(s) have nothing to do with mapping ports to boards. The format of the starting and ending port numbers in the `portMap` string can be specified with this in mind. The following are examples of the different formats that can be used to specify starting and ending port numbers:

,1, : Port table has only 1 index, make a match of this string with the single term returned as the suffix OID when the port index attribute is read.

,1.,: Port table has multiple indexes, make a match with only the first term of the returned instance id when the port index attribute is read. In this case the first term (term 1) is the port number, the additional terms are of no use.

,.1, : Port table has multiple indexes, make a match with only the last term of the returned instance id when the port index attribute is read. In this case the last term of instance id is the port number, the other term(s) are of no use in modeling the repeater.

,.1., : Port table has at least three indexes, the port number is neither the first or the last (probably the second of three indexes).

Each SP or EP specification can have multiple terms such as ,.2.1, ...

The following are examples of values for the `portMap` attribute:

```
2,1,8,3,9,16,4,17,24
```

In this example, there are three boards in the hub, boards in slot 2,3, 4. Ports numbered 1 through 8 are on board 2, ports 9 though 16 are on board 3 and ports 17 through 24 are on board 4.

```
2,.1,.2,3,.3,.10,2,.11,.14
```

In this example, there are two boards numbered 2 and 3. The port table in the repeater MIB actually has multiple indexes, but the last index is the one used to indicate the port number. In this example, port 1,2 and 11 though 14 reside on board 2, while ports 3 through 10 reside on board 3.

groupIndex_Attr (1,2)

This attribute should be set with the attribute ID of an attribute that returns an integer value that represents a group or board number. The referenced attribute should be the index attribute in the board/group table of the repeater MIB (an external, list attribute that is readable). This is the attribute that is used to discover which boards are supported by this repeater MIB.

It is possible that the referenced attribute could be a non-list attribute. and that the attribute does not have to be external. This design was developed for two reasons:

- There are some hubs that provide a separate SNMP agent for each board plugged into the hub. In these cases, the MIB is structured so that the board information in the MIB is not in a table but rather in a set of non-list attributes (the agent only has information about that single board).
- It may be advantageous to imitate the presence of a board so that the ports being modeled can be associated with a model other then the main device model. In the case of a virtual board or group, the attribute reference by `groupIndex_Attr` is an internal integer attribute set to the appropriate board number (most likely `0`).

This attribute is only used when modeling a standard repeater MIB, that is a repeater MIB whose structure is based on the existence of two tables; a group and a port table. If your repeater MIB does not have board table, or more specifically, an attribute containing the board number(s), set this attribute to `0`. See the discussion of the attributes `useMapping` and `portMap` for more details.

groupTerm (1,2)

This attribute is used in the mapping of ports to boards in a standard repeater MIB. The value of this attribute should specify the index of the port table that represents the group or board number. The port table in a standard repeater MIB typically has two indexes; the group/board number and the port number. The order of the indexes can be either “board, port”, or “port, board”. The value of this attribute defines which of the two orders is used when accessing the port table. The default value for this attribute is “board, port”.

As the port table is read from the device, each instance ID (OID suffix) returned with the port index read is examined. The term of the OID, specified by the value of this attribute, is used to determine which board

this port resides on. When a model is created to represent this port, it is placed in the appropriate association with the proper board model.

Some repeater MIBs have port tables which are not accessed by multiple indexes, but rather by a single port number. However, the MIB's port table does contain several additional attributes that provide the physical mapping of these ports to their respective boards (for example, `portPhyBoardIndex` and `portPhyPortIndex`). The difference from the standard repeater MIB port table is that the attribute containing the board number is not an index attribute. The board number does not appear in the OID suffix as the index of each port is read.

If you are modeling such a MIB, set the `groupTerm` attribute to `0`. A subsequent change is related to the `portIndex_Attr`. Instead of setting it to the true index of the table (the port number), the developer must set it to the attribute in the table which contains the board number (see `portIndex_Attr` for details).

portIndex_Attr (1,2)

The value of this attribute should be set to the attribute ID of the port index attribute (found in the port table) from the repeater MIB model type. The referenced attribute, when read from the device, should return a port number. The reference attribute must be an external, list attribute of type integer.

This attribute is used to discover each of the ports accessible through the repeater MIB. The intelligence associated with the `GnDataRelay_MF` model fragment will use the referenced attribute in a `read_next`. For each successful `read` of the attribute, it will create a port model.

As each port is discovered in the repeater MIB, a model type will be created in the database to represent that port. The model type created is a model type derived from the base model type `Port`, this model type contains the `Component` model fragment. Within the `Component` model fragment is the `Component_OID` attribute. This attribute will be set with the instance ID (suffix OID) returned when reading the port index attribute from the port table.

Some repeater MIBs have port tables which are not accessed by multiple indexes, but rather by a single port number. However, the MIB's port table does contain several additional attributes that provide the physical mapping of these ports to their respective boards (for example, `portPhyBoardIndex` and `portPhyPortIndex`). The difference between this and the standard repeater MIB port table is that the attribute containing

the board number is not an index attribute. The board number does not appear in the OID suffix as the index of each port is read.

To model the ports of such MIBs, the developer must set the `groupTerm` attribute (see above) to the value of 0. The value of the `portIndex_Attr` must then be set with handle of the attribute within the port table that is the board number. The intelligence will read this attribute and use the value (not the suffix OID) returned to determine if the port associated with this entry of the table is associated with the board being populated with ports. If the value of the attribute is the same as the board number being configured, then the intelligence will create a port model.

Note: The port model will still contain the Instance ID (`Component_OID`) produced from the suffix OID associated with the attribute read.

Note: The attribute referenced through `portIndex_Attr` is used in the discovery of port models. For each successful read, a port model with the associated Instance ID (suffix OID) is created. It is important to note which board this port is associated with, a term of the instance ID, or the value of the attribute referenced through `portIndex_Attr`. Either the term of the suffix OID will produce the board number (the `groupTerm` term) or the value read from the attribute (the `groupTerm)` is set to `0`.

portType_Attr (2)

This attribute contains the attribute ID of an attribute that returns a value which determines the type of port being reference through the repeater MIB. The referenced attribute should exist in the same table as the `portIndex_Attr`, i.e. the port table of the repeater MIB. The referenced attribute can be of any type supported by the SpectroSERVER database, usually it is either an integer or an object ID. This attribute is read by the chassis manager and used to determine the model type used to model this port in the database. If the repeater MIB does not contain a type attribute for each port, then the value of this attribute should be set to `0`.

Note: The attribute referenced by `portType_Attr` must be found in the same table of the MIB as the port index attribute. The value read from the attribute referenced by the `portType_Attr` is used in conjunction with the `portType_Map` and `portName_Map` attributes.

It is possible that the port table of the MIB being used to model the ports does not have a type attribute. However, you may still want to select

different port model types or make an icon selection based on the type of board the ports are physically located on (see the `portType_Map` and `portName_Map` attributes in the following sections).

To achieve this functionality (using board type in place of a port type attribute) you must use the board type to model ports, and you must set the `portType_Attr` with the handle of an attribute of the board model types that are being used to model the boards.

For example, much of the board modeling will be done with the `GnModule` model type, which has an attribute called `gnType` that holds the type of that board. As each board model is created, the `gnType` attribute is set with the value of the type attribute read from the chassis's slot table. By setting `portType_Attr` with the handle of the `gnType` attribute (`0x3d0032`), the port modeling intelligence can now access the board's type to be used in selecting model types or port icons.

portType_Map (2)

The value of this attribute is used when the chassis service (intelligence associated with the `GnDataRelay_MF`) needs to create port models of ports on a hub device. This attribute is used to determine which model type will be used to model each port type discovered in the hub device (see `portIndex_Attr` and `portType_Attr` above). As each port is found its type is read, and that type is used as an index value to this map to determine the model type to use when creating a port model in the SPECTRUM database.

This attribute is of the type Text String. The value of this string is read and interpreted by the chassis intelligence. The chassis intelligence that will create a map from contents of this attribute. There are two types of maps that can be specified by the user; ENUM and RULE.

ENUM Map Type

A map of type ENUM is simply an enumeration of port type and model type handle, the map string would have the following format:

```
ENUM,default_mth,pt,mth,pt,mth,pt,mth,...,pt,mth
```

Where

ENUM: A string that identifies this map type as an enumeration type.

default_mth: This is the default model type handle, this handle is used when no entry is found for the particular port type being modeled.

pt: (port type) A string representation of the value(s) that can be read from the port type attribute in the repeater MIB (`portType_Attr`).

mth (model type handle) A string representation of the model type handle for the port model type to create when a port of type PT is found in the repeater.

Note: The format used is one where each item in the string is separated by a comma and there are no intervening spaces between the comma and the value.

The following map is the default map within the `GnDataRelay_MF` model fragment. As you can see, it will create a `GnPort` model (`0x3d000b`) for each port found in the hub. The following string is the default value found in the `portType_Map` attribute:

```
ENUM,0x3d000b
```

Below are two examples. In the first example port types are integers, the second example port types are object ids.

```
ENUM,0x270015,38,0x270010,6,0x270056,18,0x270009,34,0x270023
```

In this example, the default model type to use when modeling a port is `0x270015`. If a port of type `38` is found in the hub the model type associated with the handle `0x270010` is created, if a port of type `6` is found, a model type of `0x270056` is created, etc.

```
ENUM,0xd0013,1.3.1.6.1.4.1.75.5.9,0xd0014,
1.3.1.6.1.4.1.75.5.18,0xd0038
```

RULE Map Type

The second type of port map is a RULE map. This map is based on the use of the SPECTRUM relations and the hierarchical database structure. To use this rule, the developer should have a model type created for all possible port types that can be found in the hub being modeled. The developer must set up a relation in the database that can be used to find these port model types, for example a rule such as `AcmeBoards Contains AcmePorts`. The contents of a RULE type map then instructs the chassis manager on how to find all these port model types in the database using the specified relation.

The syntax of the string used to define the RULE map is the following:

```
rule, def_mth, relation, left_mth, type_attr, group_attr,
prec_attr
```

Where:

rule: This string specifies that the map to use is of the type RULE

def_mth: This is the model type handle of the port model type that should be created when no match is found in the map for a particular port type.

relation: This is the relation handle of the rule used in mapping out the port types. For example, if you use the `Contains` relation, then the value used here is `0x10001`.

left_mth: This is the model type handle of the of the model type specified on the left side of the rule. For example, if you are modeling an ACME hub, you would setup a derivation point in the database for modeling all boards associated with the ACME hub called `ACMEBoards`. You would then use the model handle associated with `ACMEBoards` as the LEFT MTH of relation used in creating our map, i.e., the map will be built with any port model type that matches the `AcmeBoards Contains <model type>` rule.

type_attr: This is the port model type attribute id that contains a value that represents the port type. The value read from the referenced attribute is compared with the value read from the repeater MIB (see `portType_Attr`). When each port model type is created in the database, this attribute's value should be set to the value that will be returned when reading the repeater MIB.

group_attr: This is the attribute id within our ports model types that contains a value used for classifying a set of ports. This group attribute and the precedence attributes constitute a mechanism to obsolete or replace old model types. This value should specify which attribute within the port model type is used to specify the port group.

prec_attr: This is the port model type attribute id that determines the precedence of ports sharing the same group value.

Note: The GROUP ATTR and PREC ATTR arguments of the RULE specification are optional. These two parameters are needed only in the event that you have set your database up so that model types can be replaced using the meta- rule scheme.

If the rule mapping scheme is used, the setting of `portType_Map` attribute scheme could appear as follows:

```
RULE,0x3d000b,0x10001,0x750002,0x118be
```

In this example, the default model type specified is the GnPort (0x3d000b), the relation is Contains (0x10001), the model type to appear in the left side of the rule is the AcmeBoards (0x750002) model type, and the attribute Internal_Port_Type from the Port model type is used to hold the port type value within the model types used to model ports. There is no group or precedence attributes associated with the model types to be used.

altPibPrefix (1,2)

This is a text attribute that is used by the chassis manager intelligence. It contains a string of text that, when appended to a board model's name, will produce a unique string to be used by SpectroGRAPH for indexing and referencing external files (CsPib, CsIIB and CsGIB files). This is part of an alternative method used by some of the SpectroGRAPH views to display boards and ports. By default the value of this attribute is <empty>, and is only used if the developer uses this alternative method.

For a complete explanation of this attribute's use and the alternative method used to display ports and boards, see the explanation of the modulePibPrefix attribute in The GnChassis_MF Model Fragment [Page 111].

portName_Map (1,2)

This attribute, like altPibPrefix, plays a role in the alternative method used to display ports and boards. This attribute you to model ports using one model type (such as GnPort), but use a set of external files in the SpectroGRAPH to display the ports that are not related to GnPort.

In the past, SpectroGRAPH Device and DevTop views used the model type name to determine which IIB files to use to get the icon definitions for drawing the models in the view. If you modeled ports using GnPort model type, you had to use the GnPort.DIBase IIB file.

Each GnPort and GnModule model now has an attribute that is a text string. Within this text string attribute is stored an alternative model type name. For the Device and DevTop views, the contents of this attribute *can* be used instead of a model type name when indexing into a PIB file. The GnSNMPDev hub support implementation has taken advantage of this change in the views and this attribute is one of several that is used in this scheme.

The portName_Map attribute is a text attribute similar to the boardName_Map in the GnChassis_MF model fragment. It is a set of

enumerated values used to map port types to name strings. The name strings are the names you use for an alternative model type name.

The format of the string is as follows:

```
def_name,pt,pn,pt,pn,pt,pn...pn
```

Where:

def_name: This is the text string that will be used as the name of the port model type for which an entry (PT) is not found within this map. If no default name is specified (text of this attribute starts with the ‘,’ character) then the actual model type name will be used in indexing the PIB files.

pt: This is a string representation of the value returned when the port type is read from the device (via `portType_Attr`). The port type strings are the keys or index values for the map. The code reads the type from the device, converts it to a string format, then uses the string to compare against each of the PT values specified in this string. A port type (PT) entry should be provided for any type of port that can be found in the repeater MIB being modeled.

pn: (Port Name) These entries in the map string represent the names that will be used as the external file (PIB) indexes. A port name entry must be supplied for every port type entry (PT).

The following is an example of a `portName_Map` where the type value read from the device is an integer value:

```
acme_gnport,3,acme_ENETp,5,acme_TRp,4,acme_FDDIp,...9,acme_AUIp
```

In this example, the `GnPort` model type is used to model all ports found in the acme hub. However, in the acme hub device view, different icons will be used to display the different port types. To achieve this, set up the `portName_Map` so that all Ethernet ports (type 3) will be referenced as `acme_ENETp`, token ring ports (type 5) will be referred to as `acme_TRp`, etc. If these ports are to be displayed in the Device view, then the following entries would appear in the `CsPib/CsGDView.pib` file:

```
....

acme1.0/enetp.DIBase,acme_ENETp

acme1.0/trp.DIBase,acme_TRp

acme1.0/fddip.DIBase,acme_FDDIp

.....
```

Note: This is not a tutorial on the use of CsPib files. It only shows a complete example of how you can use the `portName_Map` attribute to provide an alternative means of indexing the PIB files so you will not have to create model types just to reference the specific IIB files to display your ports.

portAttachRel (2)

This attribute contains the relation handle to be used when associating ports to boards, or ports to the main device model. This attribute was added to the `GnDataRelay_MF` model fragment because the model fragment is actually used as part of two different derivation points - one to model repeaters and another to model port oriented devices (see The GnDeviceIO_MF Model Fragment [Page 128]). The default setting is the `HASPART` relation.

The GnPortUI_MF Model Fragment

The `GnPortUI_MF` model fragment is a base model type for GnSNMPDev's `GnRelayDerPt` derivation point and provides a portion of the generic hub support for GnSNMPDev. This derivation point is provided to help create application model types used to model third-party repeater MIBs. One of the significant features provided by the generic hub support is the ability to create chassis Device views for the hubs being modeled. This model fragment provides a mechanism for creating the generic Device view without creating IIB files for use by the SpectroGRAPH application.

The three attributes which make up the `GnPortUI_MF` model fragment are used in the display of the port models in the chassis Device view. Each board found in the hub is displayed in the Device view. A board is displayed with each of the associated ports found on that board.The Device view shows three pieces of data for each port; port number, a port status, and a port counter. This model fragment has attributes that are used by the Device view to display a status and counter associated with each port.

The attributes in this model fragment are read when the Device view is first created. The value of these attributes provides the SpectroGRAPH with instruction on the external device attributes to read to find the value for the port status and port counter.

These attributes also provide information on how to interpret the port status value that is read. Instead of displaying the integer value read from the port status attribute the Device view displays a text explanation of that value (off, on, linked, etc.). Because this information is provided in the

GnPortUI_MF model fragment and not in an IIB file, new IIB files do not have to be created when a new hub is modeled.

counter_Attr (1,2)

The value of this attribute is set with an attribute id from an imported MIB model type. The referenced attribute returns a counter or integer value associated with a port. The referenced attribute is found in the port table of the repeater MIB, typically the same table in the MIB from which the `portIndex_Attr` attribute was set (see The GnDataRelay_MF Model Fragment [Page 135]). This attribute should be read from the device using the same indexes as those set in the `Component_OID` attribute of each port model.

The referenced attribute will be displayed in a rate gauge icon. The attribute will be read every `5` seconds and the value is displayed as a rate of change for that time period. Typically, the attribute selected would be one that represents the number of frames or octets received or transmitted through the port.

The value of this attribute will typically be read once by SpectroGRAPH, which then uses the attribute ID to directly read the counter from the device.

status_Attr (1,2)

The value of this attribute is set with an attribute ID from an imported repeater MIB model type. The referenced attribute returns an integer value associated with some status of a port. The referenced attribute is found in the port table of the repeater MIB, typically the same table in the MIB from which the `portIndex_Attr` attribute was set (see The GnDataRelay_MF Model Fragment [Page 135]). This attribute should be read from the device using the same indexes as those set in the `Component_OID` attribute of each port model.

The referenced attribute will be displayed in the Device view with the contents of the `statusEnum_VTC` attribute (see below). Typically, the attribute selected from the MIB is an administrative or operational status found in the port table of the repeater MIB.

The value of this attribute is read once by the SpectroGRAPH which then uses the attribute ID to directly read the status from the device.

statusEnum_VTC (1,2)

The value of this text attribute is used in displaying the status of the port that is read from the `status_Attr` attribute (see above). The contents of

this attribute is an enumeration string that is typically found in an IIB file. The enumeration string is used to convert an integer status value into a text string to be displayed using a specified color. Each enumeration in the string has three values: the value read from device, text to display, color to use in displaying text (thus VTC). The following is an example of an enumeration string that could appear in `statusEnum_VTC`:

```
1,off,123,2,on,147,3,link,128,4,nop,116
```

In this example, if a status of 1 is read from the device, the port icon in the Device view will display the word `off` in red (`123`). If the status of 2 is returned, the word `on` will appear in the color green, etc.

The value of this attribute should be set when a status attribute is selected for the `status_Attr`. The information required to set this attribute is typically found in text description of the MIB. Find the entry in the MIB for the attribute referenced in `status_Attr` the MIB should describe all the possible values that can be returned when this attribute is read, and an explanation of what each value represents.

Like `counter_Attr` and `status_Attr`, the contents of this attribute is usually read only when the Device view is first opened. The value returned from reading this attribute is used to interpret and display the port status in the Device view.

Appendix I: Relations

This section describes relations that can be defined to describe relationships between model types.

In This Section

Overview [Page 150]

Lost and Found Intelligence [Page 150]

Overview

Relations define the potential ways that model types can be related to each other. `Contains`, `Manages`, and `Connects_to` are all examples of relations. Each relation has a relation handle (normally represented in hexadecimal format) that uniquely identifies it.

Meta-rules provide meaning to a relation by defining the context in which the relation should be used. A meta-rule identifies the model types that can participate in a relation. When SPECTRUM instantiates a model, the applicable relations between the model and other models are also instantiated. An instantiated relation is called an association. The association must follow the meta-rules that define the relation.

The Model Type Editor allows you to create your own relations and meta-rules for that relation. It also allows you to create meta-rules that apply to relations that have already been defined in the knowledge base. However, you can only create meta-rules for model types that have been created with your developer ID.

For more information on the definition of relations, meta-rules, and associations refer to the *SPECTRUM Concepts Guide (0467)*.

For instructions on creating relations and meta-rules, refer to the *Model Type Editor User's Guide (0659).*

Lost and Found Intelligence

SPECTRUM's lost and found intelligence monitors models to ensure that each model exists either at the top level of one of the hierarchical views (Topology, Location or Org-Chart) or within one or more container models within these views. To do this, SPECTRUM evaluates the model's

associations. A model must exist on the right side of at least one association that implements one of the following relations:

- Collects
- CollectsChassis
- CollectsConcentrator
- Contains
- HASPART
- IsAssignedTo
- IsServerTo
- Manages
- Organizes
- Owns
- Provides
- PULLED_BOARD
- Supports

For example, the `Collects` relation has a meta-rule that states that a LAN model type can collect a GnSNMPDev model type; `LAN Collects GnSNMPDev`. In this meta-rule the GnSNMPDev model type exists on the right side of the `Collects` relation. An instantiated GnSNMPDev model with a model name of `Model1` that is placed in an `LAN` container model with a model name of `MyLan` exists on the right side of a `Collects` association; `MyLan Collects Model1`.

If a model is not on the right side of an association implementing at least one of the above relations, the model will be placed in SPECTRUM's Lost and Found. For example, models are placed in the Lost and Found if the container that they were placed in was deleted, or if they are cut from the SpectroGRAPH and not pasted anywhere.

For more information on Spectrum's Lost and Found, refer to *How to Manage Your Network with SPECTRUM (1909)* and *Distributed SpectroSERVER (2770)*.

Implementing Lost and Found Intelligence for New Relations

If you create a new relation and meta-rules that allow associations between a hierarchical view and a model or a container model and a model, you may want to have your relation monitored by lost and found intelligence. This will ensure that the models that implement this relation are only sent to the lost and found when appropriate. To add lost and found intelligence to your relation, you must complete the following tasks in the Model Type Editor:

1. Create a new model type derived from the model fragment `LF_Relations`.
2. Set the `Derivable` flag on this model type to FALSE.
3. Add a new attribute of the type Relation Handle to this model type.
4. This attribute must belong to the `LF_Rels` (0x12914) attribute group.
5. The attribute's `Readable`, `Writable`, and `Shared` flags should be set to `TRUE`.
6. The attribute's `Memory` extended flag should be set to `TRUE`.
7. The value of this attribute should be set to the hexadecimal relation handle of the relation.
8. Following the steps above, create a new attribute for this model type for each of the new relations you want to be monitored by the lost and found intelligence.

Refer to the *Model Type Editor User's Guide (0659)* for specific instructions on how to create new relations, meta-rules, model types, and attributes.

Appendix J: Tutorials

The tutorials in this chapter show you how to implement the precepts described in this manual.

In This Section

Tutorial 1: Modeling Non-Data Relay MIBs [Page 153]

Tutorial 2: Modeling Port-Oriented Devices [Page 157]

Tutorial 3: Building a Hub Manager Application [Page 173]

Tutorial 1: Modeling Non-Data Relay MIBs

This chapter describes the procedures for creating an application model for a non-data relay MIB.

The simplest method of customizing GnSNMPDev is to add support for a proprietary non-data relay MIB. An example of such a MIB is the Liebert UPS Station MIB. This MIB provides management of Liebert Uninterruptible Power Supply devices. By adding management support for this MIB, you can access performance and configuration information for these devices. Also, you can perform diagnostics and battery tests, etc. by writing to some objects in this MIB.

Purpose of this Tutorial

The purpose of this tutorial is to create a SpectroGRAPH Application View model representing the Liebert Uninterruptible Power Supply MIB component of your device.

Creating the Application

To add support for this MIB, an application model type must be created that can be discovered by GnSNMPDev and can communicate with this UPS MIB. This application model type will be derived from the `GnSNMPAppDerPt` model type. The model type will have the intelligence necessary to behave as a major application but it will know nothing about the Liebert UPS Station MIB. A second model type must be created that knows about this MIB. The example will derive the second model type from the `GnSNMPMIBDerPt` model type. The second model type will then be made a

base model type of the first and the result will be a major application model type that knows about the Liebert UPS Station MIB.

Importing the Liebert UPS MIB

This section describes importing the Liebert UPS MIB (lieb_s.mib) into the SPECTRUM database by using the `GnSNMPMibDerPt` model type to create a new application model. Follow this procedure:

1. In the Model Type Editor, select **Find Model Type...** from the **File** menu.
2. Enter `GnSNMPMibDerPt` in the **Name or Handle** field.
3. Single-click the **Apply** button to bring up the `GnSNMPMibDerPt` model type.
4. Double-click the `GnSNMPMibDerPt` model type to select it as the current model type.

Create a new model type that will eventually contain the MIB.

5. Select **New Model Type** from the **File** menu and name the new model type `LiebertUPSMib`. The name must not exceed `15` characters. The Mib suffix easily identifies the model type as one containing an imported MIB.
6. Click **OK**.
7. Select **Import MIB File...** from the **File** menu.
8. In the **Selection** field, enter the directory path for the Liebert UPS MIB file (lieb_s.mib) including the filename.
9. Enter the correct value in the **SMI Path** field (`1.3.6.1.4.1`).
10. Click **OK**.

If you do not see any error messages and the Import MIB window closes, the MIB has been imported.

Creating the Application Model Type

This section describes creating a Liebert UPS application model type using the `GnSNMPAppDerPt` derivation point. The new application model type will be used to model the Liebert UPS MIB (lieb_s.mib). Follow this procedure:

1. In the Model Type Editor, select **Find Model Type...** from the **File** menu.

2. Enter `GnSNMPAppDerPt` in the **Name or Handle** field.
3. Single-click the **Apply** button to bring up the `GnSNMPAppDerPt` model type.
4. Double-click the `GnSNMPAppDerPt` model type to select it as the current model type.

From this point in the SPECTRUM database, create a new model type that will be used to model the Liebert UPS application.

5. Select **New Model Type** from the **File** menu and name the new model type `LiebertUPSApp`. The App suffix easily identifies the new model type as an application based on the Liebert UPS MIB (lieb_s.mib).
6. Click **OK**.

The new `LiebertUPSApp` model type should now be the current model type in the Model Type Editor. The `LiebertUPSMib` model type should now be added as a base model type to the `LiebertUPSApp` model type.

7. Select **Add Base Model Type** from the **Edit** menu and add the `LiebertUPSMib` model type as a base model type for `LiebertUPSApp`.
8. Click **Apply** and then **OK**.

The `LiebertUPSApp` model type should now contain two base model types: `GnSNMPAppDerPt` and the `LiebertUPSMib` model types.

Setting the default_attr_list Attribute

The new Liebert UPS application model will appear in the Application View as one of many application model types when you model your device with GnSNMPDev. For SpectroSERVER to create the new application model in the Application View, two modifications must be made: setting the `default_attr_list` attribute and making the new model type instantiable. The `default_attr_list` attribute is set using attribute IDs of objects that will uniquely identify the Liebert UPS MIB (lieb_s.mib). Follow these steps:

1. In the `LiebertUPSApp` model type's **Examine Attributes View**, find the following attributes and note their attribute IDs:

   `lcUpsIdentManufacturer`

   `lcUpsBatTimeRemaining`

   `lcUpsInputFrequency`

2. Scroll through the `Gen_Create_MF` model fragment until you find the `default_attr_list` attribute.

   ```
   CSCR Gen_Create_MF default_attr_list
   ```

3. Double-click the **Values** field for `default_attr_list`.
4. Set the value of the `default_attr_list` using the attribute IDs from Step 1.
   a. For each attribute ID, click the **Add** button. Do not enter any value; just click the **OK** button.
   b. Click **OK**.
   c. Double click on the empty field in the Values column that corresponds to the OID index entry you just made.
   d. Type the attribute ID and click **OK**.
   e. Repeat these steps for each attribute ID.
5. Set the `Model_Group` attribute to the decimal value of the `LiebertUPSApp` model's model type handle.

   Note: It is important to make sure that the value of `Model_Group` is set appropriately. If `Model_Group` is set to `0`, SPECTRUM will only use the `default_attr` attribute to identify the application model type that represents the MIB's functionality.

6. Make the `LiebertUPSApp` model type instantiable by clicking the **Instantiable** button and selecting **Save All Changes** from the **File** menu.

The GnSNMPDev model will now discover the new Liebert UPS major application by determining if an attribute that is characteristic of the new MIB application exists on the device. The application will be discovered if and only if your device contains one or more of the objects listed in the `default_attr_list`.

Tutorial 2: Modeling Port-Oriented Devices

This tutorial describes creating and configuring a model type that will be used to model a port-oriented device's ports. Additionally, it includes a section on testing the port model type and a checklist of possible modeling errors.

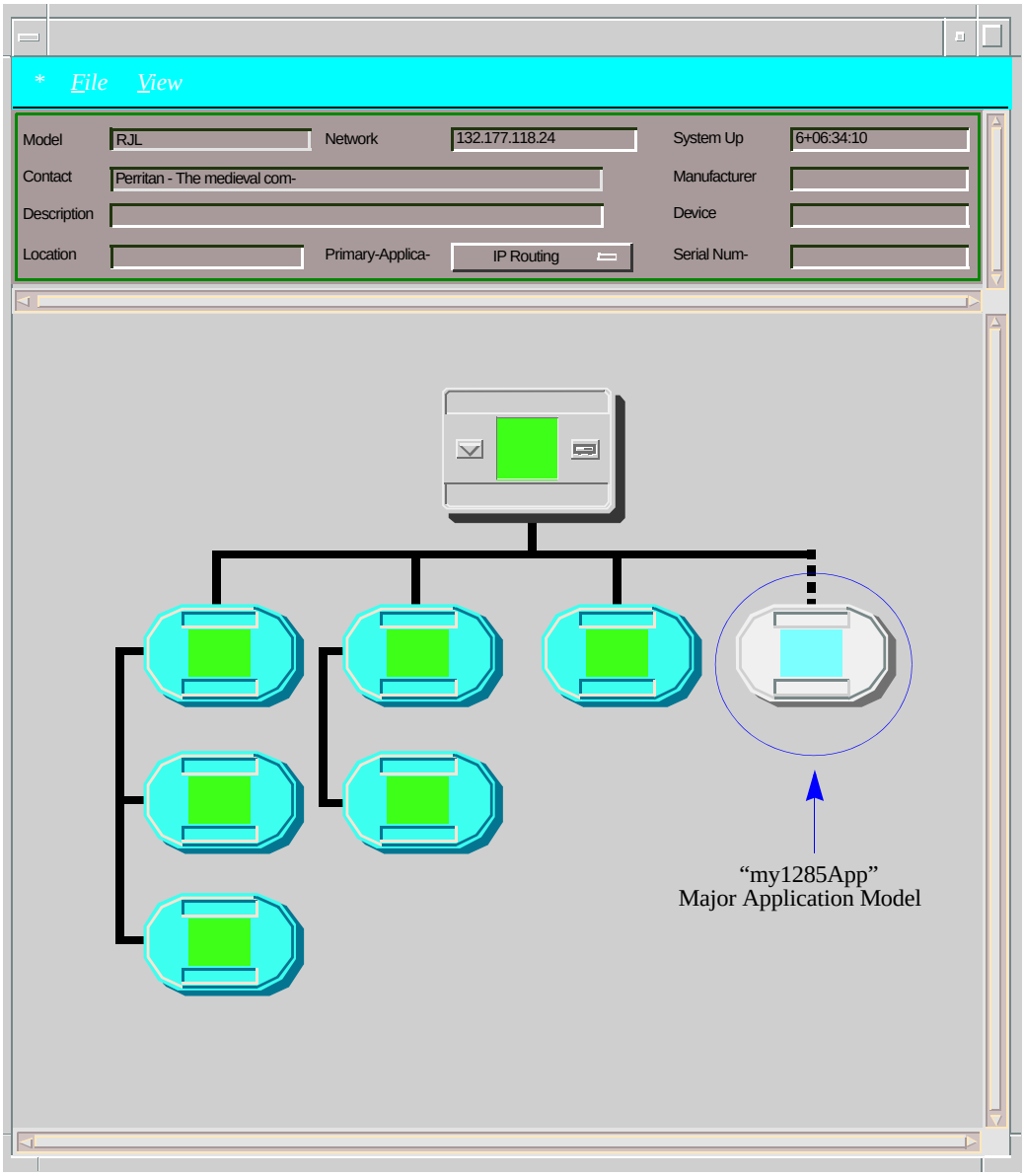
Port-oriented devices are devices with ports that are not included in the MIB-II (rfc1213) Interface Table. This would include such devices as terminal servers or switches. Ports are associated directly to the device model with no board model present. The type of port to be modeled determines the MIB to be used. In this example, FDDI ports are modeled using the FDDI MIB. This MIB is supported by a considerable number of devices any one of which can be modeled in the following exercise.

Purpose of this Tutorial

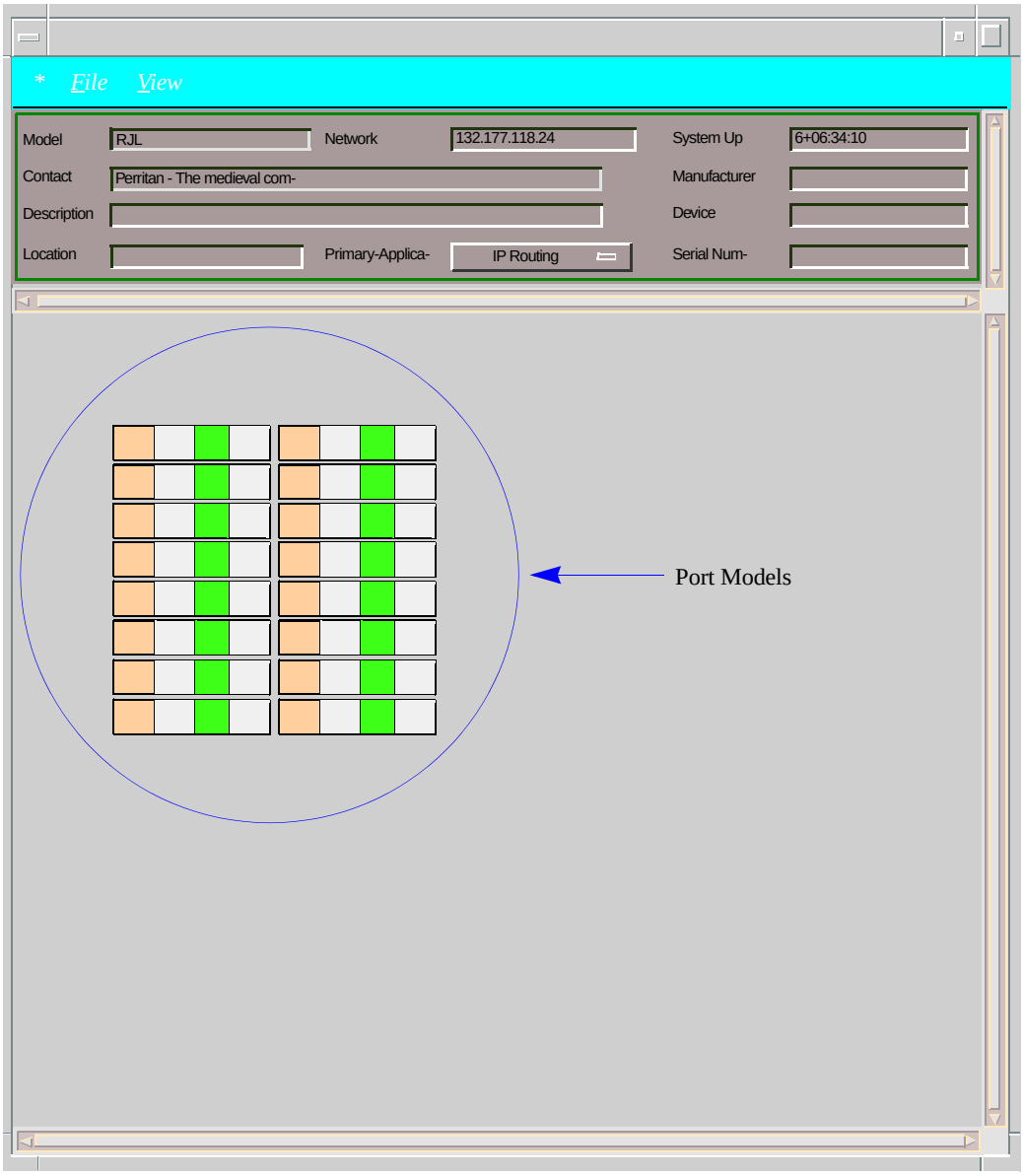
The purpose of this tutorial is to create the following SpectroGRAPH models and views for your device:

- An Application View model representing the port component of your device (Port-Oriented Device Application View Model [Page 158]).
- A Device View showing the ports associated with your device and the status and activity of each port (Port-Oriented Device View [Page 159]).
- A Device Topology View showing the ports associated with your device, the status and activity of each port, and user-resolved port connections (Port-Oriented Device Topology View [Page 160]).

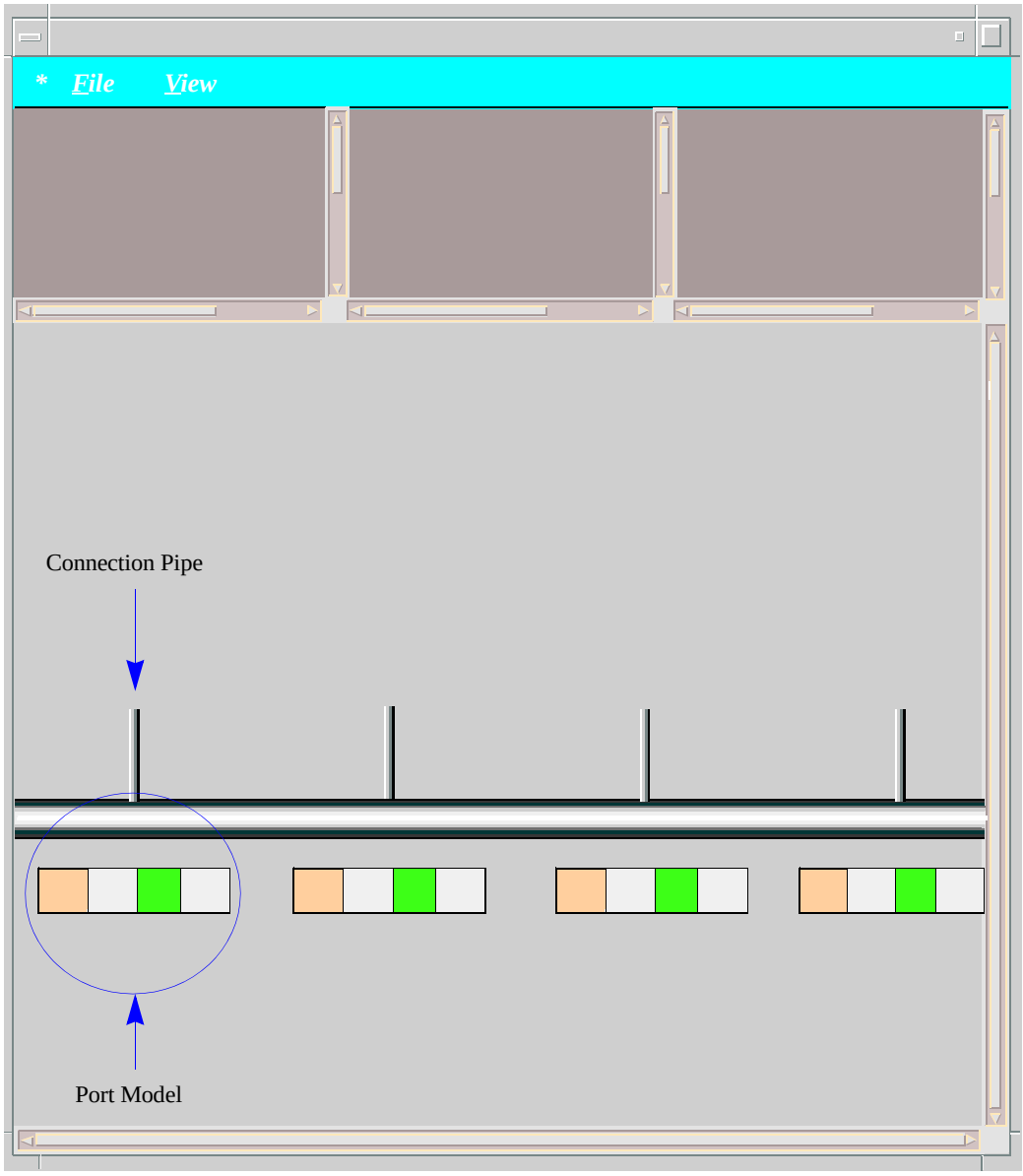
Port-Oriented Device Application View Model



Port-Oriented Device View



Port-Oriented Device Topology View



Before You Begin

There are two versions of the FDDI MIB that have been imported into the SpectroSERVER database. The following table lists their rfc numbers and corresponding model type names.

MIB	Model Type Name
rfc1285	FDDIMib
rfc1512	FDDIMib1512

You should determine which version of this MIB is supported by the device you will be modeling. To verify this, use an SNMP tool (such as `snmpget`) to query one of the following OIDs from the device supporting the FDDI MIB. This OID is the port number attribute which defines the number of entries in that MIB's port table.

```
snmpget <IP address> <community name> . <OID>
```

The following table lists the OIDs and corresponding FDDI MIBs and SPECTRUM model types.

OID	MIB and Model Type
1.3.6.1.2.1.10.15.4.1.0	rfc1285 (FDDIMib)
1.3.6.1.2.1.10.15.73.5.1.0	rfc1512 (FDDIMib1512)

Note: This exercise uses the rfc1285 MIB as the example. If your device supports the rfc1512 MIB, there will be differences in the attributes names and attribute IDs you will need to model your device. You should note these differences as you go through this exercise.

Gathering MIB Information

You will need to select a set of attributes in the FDDIMib model type that will be used to model the ports, as follows:

1. Select **Examine Attributes** from the **File** menu.

2. In the FDDIMib model type's **Examine Attributes View**, scroll down the **Attribute Names** list until you find the attributes listed in this section.
3. As you find each of the following attributes, record the attribute ID or handle that was assigned to the attribute when it was imported into the `FDDIMib` model type. The attribute name as it appears in the `FDDIMib` model type is displayed in bold below each attribute's description.

The following attributes are used to discover, model, and attach port models referenced through the rfc1285 MIB.

- **Discovery Attribute**

  An external attribute that will be part of the application discovery process. A mandatory, non-list, attribute is preferable. When you model your device with GnSNMPDev, SPECTRUM intelligence will discover the new MIB application by determining if an attribute that is characteristic of the new MIB application exists on the device. The GnSNMPDev model looks for the attribute specified in `default_attr` as described later in this chapter.

  `CSCRFDDIMibsnmpFddiPORTNumber23012e`

- **Port Index**

  An external attribute that returns an integer value when read. The integer value represents the port number. The attribute ID assigned to port index will be used to set the value of `portIndex_Attr` attribute found in the `GnDataRelay_MF` model fragment.

  `CSCRFDDIMibsnmpFddiPORTIndex230132`

The following attributes control the display of the port models in the Chassis Device and Device Topology Views. The Chassis Device and Device Topology Views display a logical representation of each port. The icon used to display a port in each of these views requires two pieces of information: a counter read from the port and a status read from the port. Attributes from the port table, referenced by the port index discussed previously, should be used to ensure that the same instance ID is used to read the status and counter for the same port. In the port table, find the following attributes:

- **Counter**

  An external attribute that has the object type of counter, integer, or gauge. When read, it will return some accumulating value associated

with each port (e.g. number of frames, packets or octets received or transmitted, etc.). The attribute ID assigned to counter will be used to set the value of the `counter_Attr` attribute found in the `GnPortUI_MF` model fragment.

`CSCRFDDIMibsnmpFddiPORTLemCts230141`

- **Status**

  An external attribute that has the object type of integer. When read, it will return some operational or administrative status information for the port. The attribute ID assigned to status will be used to set the value of the `status_Attr` attribute found in the `GnPortUI_MF` model fragment.

  `CSCRFDDIMibsnmpFddiPORTConnectState230144`

- **Status Value**

  Status value refers to displaying the value read from `status_Attr`. The status value does not have an attribute ID. In the rfc1285 MIB, the `snmpFddiPORTConnectState` object may look something like this:

  ```
  snmpFddiPORTConnectState OBJECT-TYPE

  SYNTAX INTEGER {

  disabled (1),

  connecting (2),

  standby (3),

  active (4)

  }
  ```

  Reading this status will return a value of 1, 2, 3, or 4. This information will be used to set the `statusEnum_VTC` attribute in the `GnPortUI_MF` model fragment. The `statusEnum_VTC` attribute is a text string that provides an enumeration of Value, Text, and Color used in displaying the status information for the port icon in the `statusEnum_VTC` Chassis Device and Device Topology Views.

  After you have found the attribute IDs, select **Close Attributes** to exit the **Examine Attributes** View.

Building the Application Model Type

This section describes creating and configuring an application model that will be used to model the device' s ports. Additionally, it includes a section on testing the model at this point and a checklist of possible modeling errors.

Creating an Application Model Type

This section describes creating an application model type using the GnDevIODerPt derivation point. The new application model type will be used to model the device's ports. Follow this procedure:

1. In the Model Type Editor, select **Find Model Type...** from the **File** menu.
2. Enter `GnDevIODerPt` in the **Name or Handle** field.
3. Single-click the **Apply** button to bring up the `GnDevIODerPt` model type.
4. Double-click the `GnDevIODerPt` model type to select it as the current model type.

From this point in the SPECTRUM database, create a new model type that will be used to model the application.

5. Select **New Model Type** from the **File** menu and name the new model type. For discussion, name the new application `my1285App`. The rfc number and App suffix easily identifies the new model type as an application based on the rfc1285 MIB.
6. Click **OK**.

The new `my1285App` model type should now be the current model type in the Model Type Editor. The `FDDIMib` model type should now be added as a base model type to the `my1285App` model type.

7. Select **Add Base Model Type** from the **Edit** menu and add the example MIB model type as a base model type for `my1285App`.
8. Click **Apply** and then **OK**.

The `my1285App` model type should now contain two base model types: GnDevIODerPt and the `FDDIMib` model type.

Setting Up the Application Model Type

The new application model will appear in the Application View as one of many application model types when you model your device with GnSNMPDev. For SpectroSERVER to create the new application model in the Application View, several modifications must be made: setting a text string to be displayed on the Application Icon, setting the `default_attr` attribute, and making the new model type instantiable.

Naming the Port Model

In the Application View, the port application icon will appear with the `my1285App` model type name displayed on zone (c) of the icon (refer to Figure A-1). Zone (a) of the icon can be used to optionally display a user-defined model name. To add a user-defined name to the port model, follow this procedure:

1. In the `my1285App` model type's **Examine Attributes View**, scroll through the **Attribute Names** list until you find the following attribute:

   ```
   CS       Named_EntityModel_Name
   ```

2. Double-click the **Values** field for the `Model_Name` attribute to access the **Alter Value** dialog box.
3. In the **Alter Value** dialog box, enter the name for your port application model (e.g. FDDI Ports).
4. Click **OK**.
5. Select **Save All Changes** from the **File** menu.

Setting the default_attr Attribute

The GnSNMPDev model discovers the new MIB application by determining if an attribute that is characteristic of the new MIB application exists on the device. The GnSNMPDev model looks for the attribute specified in `default_attr`. To set the `default_attr` attribute, follow this procedure:

1. In the `my1285App` model type's Examine Attributes View, scroll through the `Gen_Create_MF` model fragment until you find the `default_attr` attribute.

   ```
   CSCR     Gen_Create_MFdefault_attr
   ```

2. Double-click the **Values** field for `default_attr`.

3. Set the attribute value to the attribute ID of the Discovery Attribute found earlier (23012e).
4. Make the `my1285App` model type instantiable by clicking the **Instantiable** button and selecting **Save All Changes** from the **File** menu.

Adding the GnPortUI_MF Model Fragment

The GnPortUI_MF should now be added as a base model type for the `my1285App` application model type. This will eliminate the need to do any external file work in order to have the port icons displayed in the Device and Device Topology Views. To add the `GnPortUI_MF` model fragment as a base model type, do the following:

1. Select **Add Base Model Type** from the **Edit** menu and add the `GnPortUI_MF` as a base model type for `my1285App`.
2. Click **Apply** and then **OK**.

The `my1285App` model type should now contain three base model types: GnDevIODerPt, the `FDDIMib` MIB model type, and the `GnPortUI_MF` model fragment.

Defining Device Configuration Management

The device you are modeling may be configurable, i.e., the number of ports can change after the device has been modeled. The model fragment's intelligence determines configurability from the setting of attributes associated with the `GnDeviceIO_MF` model fragment. The intelligence can then periodically check the device to determine whether the configuration has changed and update the corresponding SPECTRUM model to reflect this change. Two attributes control this process.

- **configurable**

  The setting of this attribute will inform the model fragment's intelligence as to whether or not the device is configurable. If this attribute is set to `FALSE` (the default), the device's configuration is not checked. If this attribute is set to `TRUE`, the intelligence will periodically check the device's configuration and compare it to the model of the device in the SpectroSERVER database. How often the configuration is checked is determined by the next attribute.

- **configCycle**

This attribute is an integer which determines the number of seconds between configuration test intervals. This attribute is used only if the configurable attribute is set to `TRUE`. If the `configCycle` attribute value is set to `0` (the default), the configuration is checked only when SpectroSERVER is first started or contact with the device is lost and then re-established. This setting would be useful for devices that need to be powered down to change the port configuration. If this attribute's value is not zero, this indicates, in seconds, how often the model fragment's intelligence will check the device's configuration against the corresponding model of the device in the SpectroSERVER database. This setting would be useful for a device that can be reconfigured without being powered down.

If the device you are modeling is configurable, do the following:

1. In the `my1285App` model type's **Examine Attributes View**, scroll down the Attribute Names list until you find attributes associated with the `GnDeviceIO_MF` model fragment.
2. Find the configurable attribute.

   ```
   SNMP     GnDeviceIO_MFconfigurable
   ```

3. Double-click the **Values** field for configurable.
4. Set the attribute value to `TRUE`.
5. Click **OK**.
6. Find the configCycle attribute.

   ```
   SNMP     GnDeviceIO_MFconfigCycle
   ```

7. Double-click the **Values** field for configCycle.
8. In the **Alter Value** dialog box, specify the number of seconds for the configuration test interval. For example, `120` (every 2 minutes).
9. Click **OK**.

Modeling the Ports

The `GnDataRelay_MF` model fragment is used to describe the ports found on the device. This model fragment's attributes and intelligence are used by GnSNMPDev to discover, model, and attach port models referenced through the MIB. Follow this procedure:

1. In the my1285 Application model type's **Examine Attributes** view, scroll down the **Attribute Names** list until you find attributes associated with the GnDataRelay_MF model fragment.
2. Find the portIndex_Attr attribute.

   SNMP GnDataRelay_MFportIndex_Attr

3. Double-click the **Values** field for portIndex_Attr.
4. In the **Alter Value** dialog box, enter the Attribute ID for the snmpFddiPORTIndex (230132) that was gathered from the corresponding FDDIMib model type (refer to the "Gathering MIB Information" section).
5. Click **OK**.

The next two attributes, altPibPrefix and portGroupMth, must be set in order to adhere to the modeling scheme for GnSNMPDev.

- **altPibPrefix**

  This attribute instructs the GnDataRelay_MF model fragment's intelligence to create an intermediary module/board model to which the port models will be associated. Setting the attribute informs the model fragment's intelligence to create a board model, attach the board to GnSNMPDev and attach the ports to the board. The board model is transparent to the user.

- **portGroupMth**

  This attribute instructs the GnDeviceIO_MF as to what type of board model to create. In this case, the model type handle of 0x3d000a will be specified. This is the model type handle of GnModule.

To set the altPibPrefix attribute, do the following:

1. In the rfc1285 Application model type's **Examine Attributes View**, scroll down the **Attribute Names** list until you find attributes associated with the GnDataRelay_MF model fragment.
2. Find the altPibPrefix attribute.

   SNMP GnDataRelay_MFaltPibPrefix

3. Double-click the **Values** field for altPibPrefix.
4. In the **Alter Value** dialog box, enter GnInvMd.
5. Click **OK**.

To set the portGroupMth attribute, do the following:

1. In the rfc1285 Application model type's **Examine Attributes View**, scroll down the **Attribute Names** list until you find attributes associated with the GnDeviceIO_MF model fragment.
2. Find the portGroupMth attribute.

   SNMP GnDeviceIO_MFportGroupMth

3. Double-click the **Values** field for portGroupMth.
4. In the **Alter Value** dialog box, enter 0x3d000a.
5. Click **OK.**

Defining the Port Display

The final attribute modifications control the display of the port models in the Chassis Device and Device Topology Views. This requires setting three attributes in the GnPortUI_MF model fragment, as follows:

1. In the my1285 Application model type's **Examine Attributes** view, scroll down the Attribute Names list until you find attributes associated with the GnPortUI_MF model fragment. For example:

   SNMP GnPortUI_MFGnPortUIMF

The following attributes in the GnPortUI_MF model fragment allow SpectroGRAPH to display status information for each port which exist on the boards being modeled.

- counter_Attr
- status_Attr
- statusEnum_VTC

2. In the Alter Value dialog box, enter the appropriate Attribute ID for the counter_Attr and status_Attr attributes, that was gathered from the corresponding FDDIMib model type attributes and click OK. Table 3 references these attributes.

GnPortUI_MF Attribute	rfc1285Mib Attribute	Attribute ID
counter_Attr	snmpFddiPORTLemCts	230141
status_Attr	snmpFddiPORTConnectState	230144

The statusEnum_VTC attribute is a text string that will read the status of the port from the status_Attr attribute and display the status in a readable format on the port icon in the Chassis Device and Device Topology Views. The statusEnum_VTC attribute maps the value read from the port into a text string to be displayed. Each complete entry in the string has three values: Value read from device, Text to display, and Color to use in displaying text. Every section of the string is separated by a comma (,):

"value,text,color,value,text,color,value,text,color,value,text,color"

To construct the statusEnum_VTC text string, do the following:

1. Double-click the **Values** field for the statusEnum_VTC attribute to access its **Alter Value** dialog box.
2. Add the first Value read from device from the snmpFddiPORTConnectState MIB object (1) and add a comma (,).

```
snmpFddiPORTConnectState OBJECT-TYPE

   SYNTAX INTEGER {

               disabled (1),

               connecting (2),

               standby (3),

               active (4)

         }
```

3. Abbreviate the Text to display associated with the previous Value read from device to 3 characters or less (e.g. operational could be abbreviated to Opr) and add a comma (,).
4. Add the RGB color that you want associated with the Text to display (e.g. 121 = red, 123 = green, 128 = yellow, 154 = blue) as the Color to use in displaying text and add a comma (,).
5. Repeat steps 2 through 4 for the remaining values that can be read from the snmpFddiPORTConnectState MIB object (2, 3, and 4).

The resulting statusEnum_VTC text string could look something like this example:

```
1,Dis,121,2,Con,154,3,Stb,128,4,Act,123
```

6. Select **Save All Changes** from the **File** menu.

7. Exit the Model Type Editor.
8. Start SPECTRUM and create the device model.

The port icons in the Chassis Device and Device Topology Views will now display the port status as "Dis" (disabled) with a red background color when a 1 is read, "Con" (connecting) with a blue background color when a 2 is read, "Stb" (standby) with a blue background color when a 3 is read, and "Act" (active) with a green background color when a 4 is read.

Testing the Port Application Model

Before proceeding further, the new port application model should be modeled in SPECTRUM to make sure that the application was built correctly up to this point. The procedure for testing the port application is to model the device with the GnSNMPDev model type and then examine the Application View, the Device View, and the Device Topology View as follows:

1. Start SpectroSERVER and SpectroGRAPH.
2. Navigate to a **Topology View**.
3. Select **Edit** from the **File** menu.
4. Select the **New Model...** option from the **Edit** menu to access the **Add Options** dialog box.
5. Select the GnSNMPDev model type from the **Add Options** dialog box and click **OK**. The corresponding **Creation** dialog box is displayed. Fill in the following parameters:

   - **Network Address**

     Enter the Internet Protocol (IP) Address assigned to the device

   - **Community Name**

     Enter the Community Name that has been assigned locally to this device. The default Community Name value is `public`.

6. After entering the icon parameter information, click **OK**. The icon representing the device appears at the top of the window.
7. Return to **View** mode.
8. Highlight the icon and select **Icon Subviews** from the **View** menu.
9. Select **Application** from the **Icon Subviews** menu to access the **Application View**.

If the port application does not appear in the **Application** view, refer to the next section.

If the port application does appear in the **Application** view, exit the **Application** view and access the **Chassis Device** view, as follows:

1. Highlight the icon and select **Icon Subviews** from the **View** menu.
2. Select **Device** from the **Icon Subviews** menu.
3. Select **Chassis** from the Device menu.

If, after a minute, the **Chassis** selection remains grayed-out, then the intelligence did not create any port models.

The **Chassis Device** view provides a logical representation of the ports on the device being modeled. At this point in building the port application, a model of each port should be displayed with the correct port number. Exit the **Chassis Device** view and access the **Chassis Device Topology** view, as follows:

1. Highlight the icon and select **Icon Subviews** from the **View** menu.
2. Select **DevTop** from the **Icon Subviews** menu.
3. Select **Chassis** from the **DevTop** menu.

You should see each of your ports at the bottom of the **Chassis Device Topology** view window along with the correct port number.

1. Navigate to the SPECTRUM view contained your device model.
2. Select **Edit** from the **File** menu.
3. Highlight the device model and select **Destroy** from the **Edit** menu to destroy the model.
4. Exit SpectroGRAPH and shut down SpectroSERVER.

If the Test Fails

If the application you built did not appear as a model in the **Application** view, check the following parameters:

- Make sure that the IP address and community name for the device being modeled are correct.
- Make sure the `Instantiable` flag is set for the application model type.
- Check the setting of the `default_attr` attribute in the application model type.

- Check the OID Prefix for the MIB model type to ensure that the attribute contains the correct OID address for the device being modeled.

Tutorial 3: Building a Hub Manager Application

This chapter describes building a hub manager application through creating an application model type. Additionally, it describes testing the application, labeling the hub boards, and modeling the hub ports.

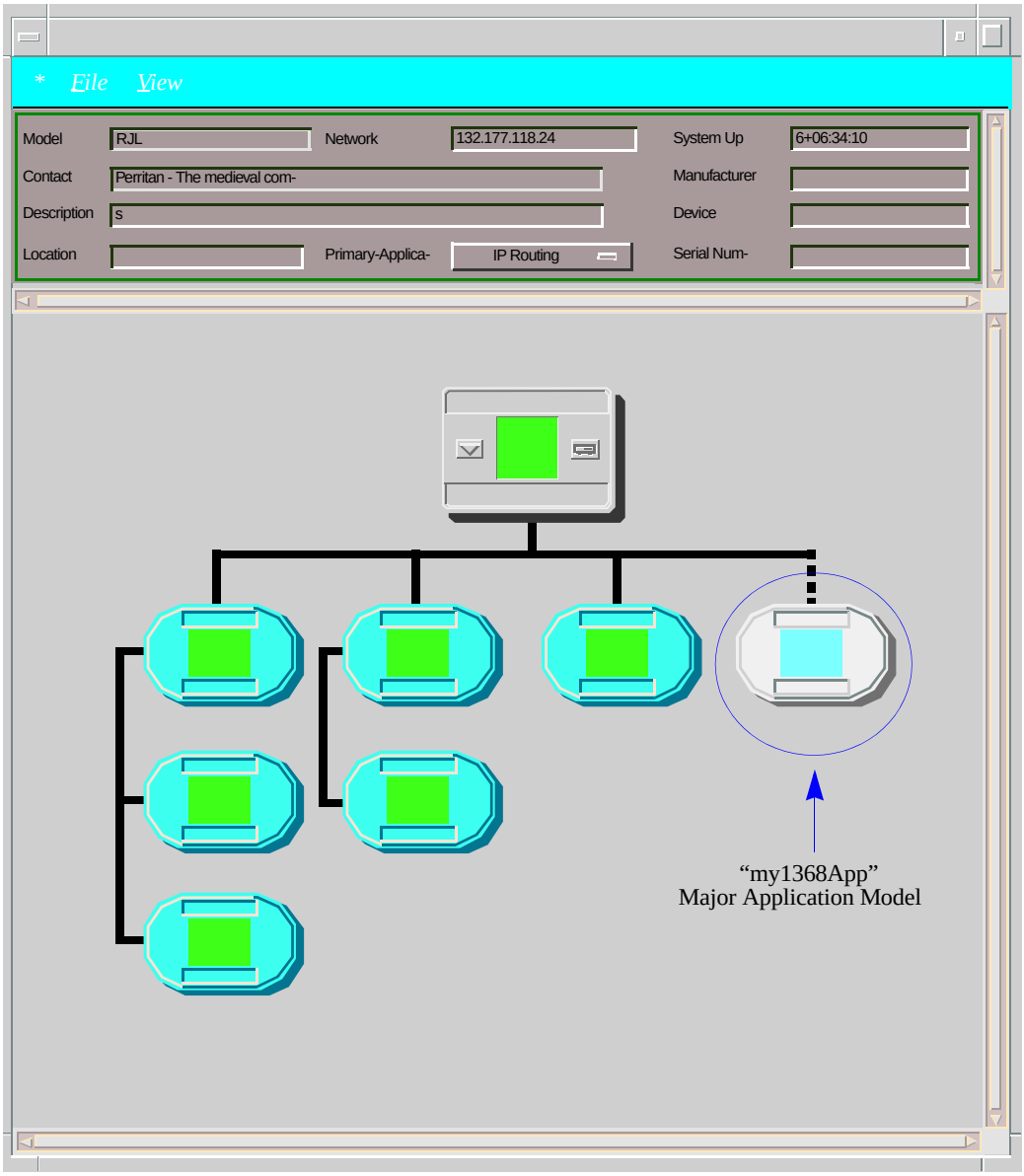
Modeling a hub manager application involves the creation of board and port models by using a specific derivation point for that purpose. Creating an application model type is basically mapping the structure of your hub/repeater MIB into the appropriate model fragment. The first part of this procedure is to find the appropriate attributes required by the modeling intelligence.

Purpose of this Tutorial

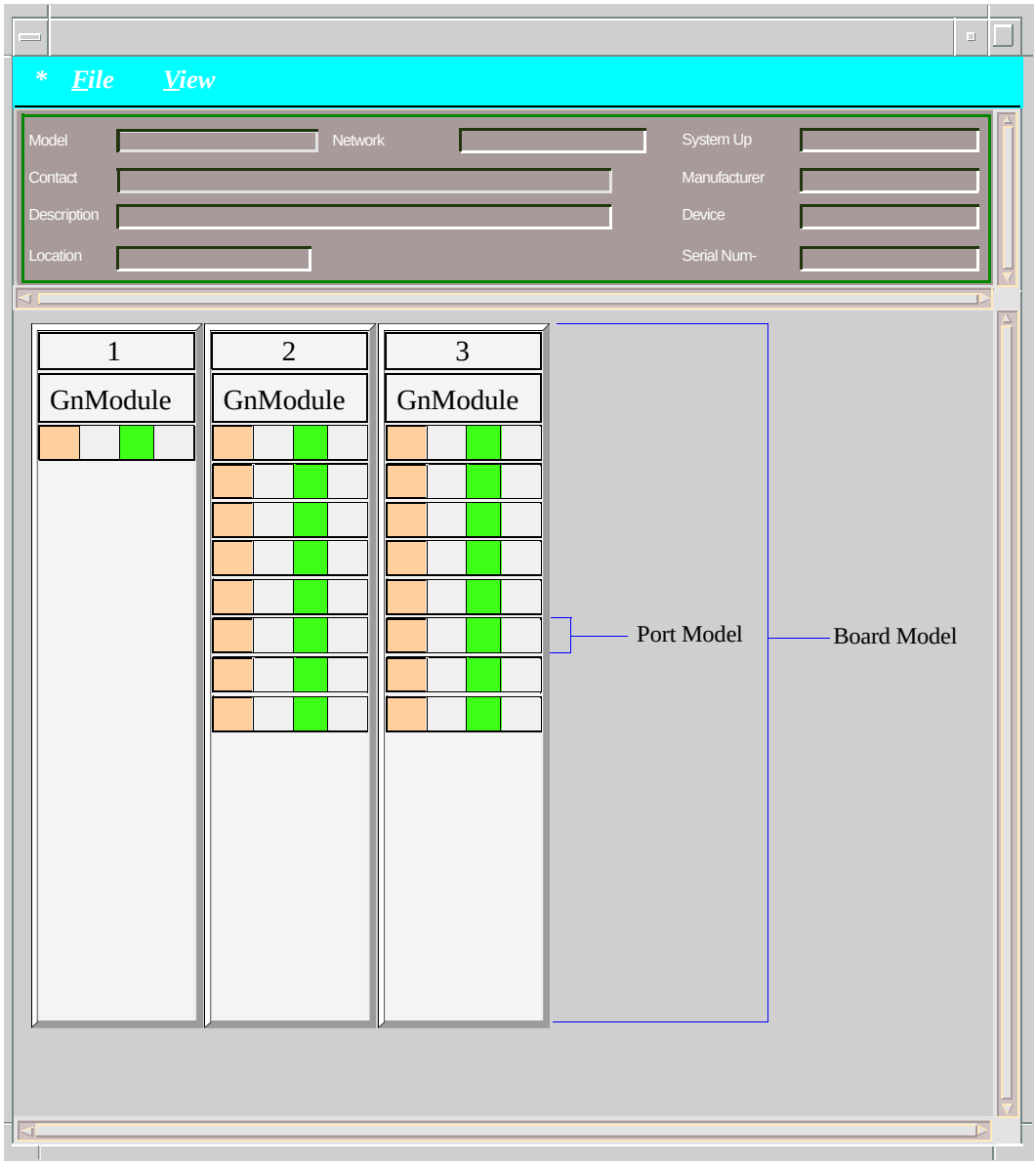
This tutorial provides board and port modeling instruction for devices that support the IETF rfc1368 Repeater MIB. The purpose of this tutorial is to create the following SpectroGRAPH models and views for your device:

- An **Application** view model representing the repeating component of your device.
- A **Chassis Device** view showing the boards and ports associated with your device and the status and activity of each port.
- A **Chassis Device Topology** view showing the ports associated with your device, the status and activity of each port, and user-resolved port connections.

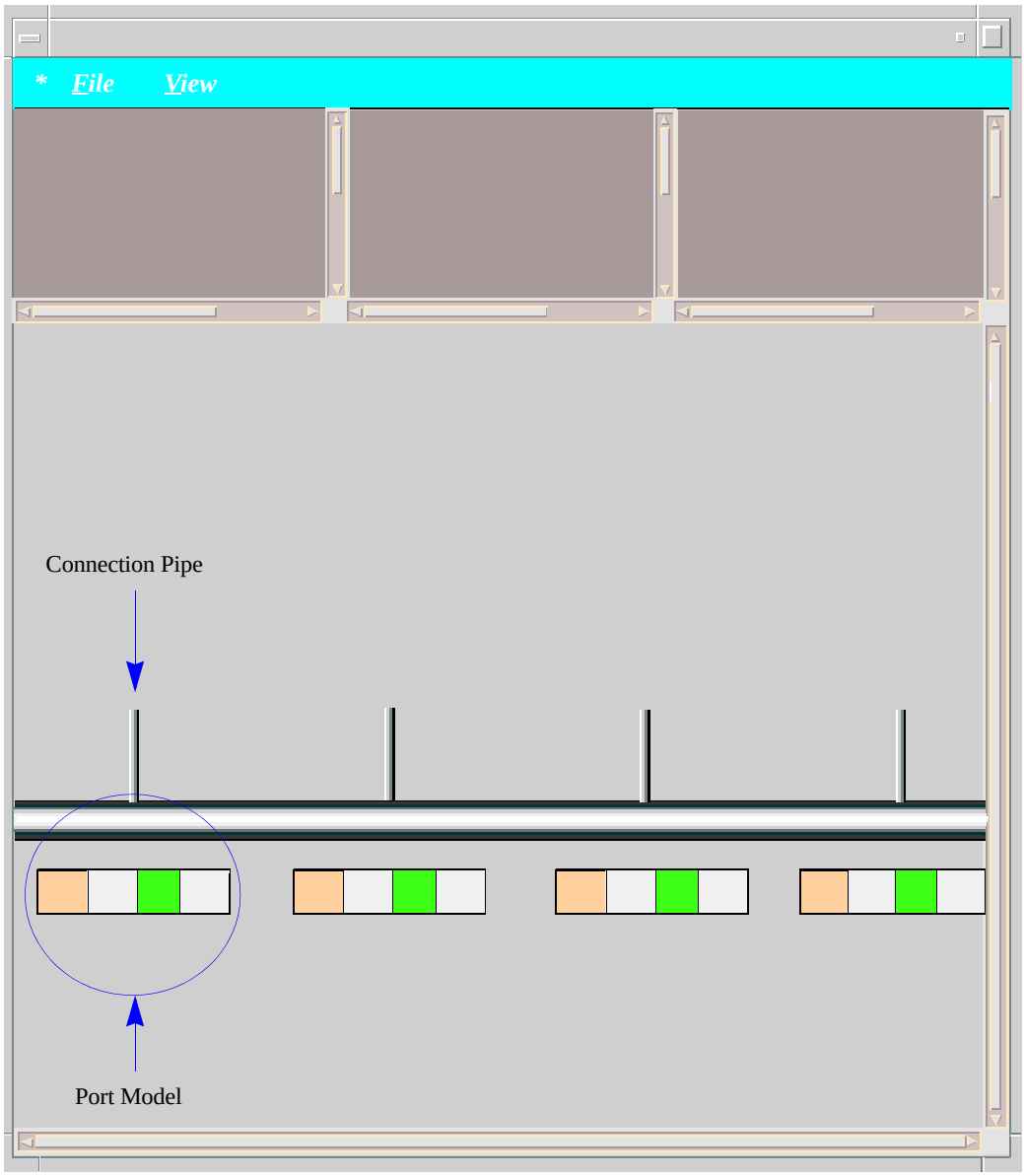
Hub Manager Application View Model



Hub Manager Chassis Device View



Hub Manager Chassis Device Topology View



Gathering MIB Information

You will need to select a set of attributes in the `rfc1368Mib` model type that will be used to model the hub manager application, as follows:

1. Select **Examine Attributes** from the **File** menu.
2. In the `rfc1368Mib` model type's **Examine Attributes View,** scroll down the **Attribute Names** list until you find the attributes listed in the following sections.
3. As you find each of the following attributes, record the attribute ID or handle that was assigned to the attribute when it was imported into the `rfc1368Mib` model type. The attribute name as it appears in the `rfc1368Mib` model type is displayed in bold below each attribute's description.

- **Discovery Attribute**

  An external attribute that will be part of the application discovery process. A mandatory, non-list, attribute is preferable. When you model your device with GnSNMPDev, SPECTRUM intelligence will discover the new MIB application by determining if an attribute that is characteristic of the new MIB application exists on the device. The GnSNMPDev model looks for the attribute specified in `default_attr` as described later in this chapter.

  ```
  CSRF rfc1368Mib rptrGroupCapacity c4000e
  ```

Chassis Information

The chassis element of the hub is modeled using the `GnChassis_MF` model fragment. This model fragment is used to define the physical nature of the chassis itself (e.g. number of slots, location of boards, types of boards, names of boards, etc.). In the hub/repeater MIB, find the slot, board, or group table. In the table, find the following attributes:

- **Board Index**

  An internal or external attribute that returns a unique value when read. The integer value represents the board number. The attribute ID assigned to Board Index will be used to set the value of `boardIndex_Attr` attribute found in the `GnChassis_MF` model fragment. The `rfc1368` MIB uses a group table.

  ```
  CSRF rfc1368Mib rptrGroupIndex c40016
  ```

- **Board Type**

An external internal or external attribute that returns an integer value when read. The unique value designates the board type found in this slot of the chassis for this manufacturer. This attribute is typically an integer or an Object ID and is usually found in the same table as the Board Index but may be in a different table as long as both tables are indexed the same way. The attribute ID assigned to Board Type will be used to set the value of `boardType_Attr` attribute found in the `GnChassis_MF` model fragment. The rfc1368 MIB uses an Object ID for this attribute.

```
CSRF rfc1368Mib rptrGroupObjectID c40018
```

- **Board Name**

  An external attribute that is a text string or octet string. This attribute may contain just the manufacturer's assigned board name or a more complete description of the board such as the firmware revision level or a functional description. If a full description is returned, then the attribute typically has the suffix Descr in the attribute name (e.g. moduleDescr). The attribute ID assigned to Board Name will be used to set the value of `boardName_Attr` attribute found in the `GnChassis_MF` model fragment.

  ```
  CSRF rfc1368Mib rptrGroupDescr c40017
  ```

Repeater Information

The repeater element of the hub is modeled using the `GnDataRelay_MF` model fragment. This model fragment is used to describe the ports found on the boards in the hub chassis and contains the information necessary to determine which ports belong on which boards. The repeater is defined in the MIB using a group table and a port table. The group table usually has a single index which corresponds to the board number to which this group is assigned (typically, all the ports within a group reside on a single physical board). The port table will have two indexes: the respective group number and a port number. This table provides the physical mapping of port to board. In the group and port tables, find the following attributes:

- **Group Index**

  An internal or external attribute that returns an integer value when read from the device. The integer value represents the group number. By convention, the group number may correlate to the board number. In our example, the Group Index is the same

as the Board Index described previously. The attribute ID assigned to the Group Index will be used to set the value of `groupIndex_Attr` attribute found in the `GnDataRelay_MF` model fragment.

```
CSRF rfc1368Mib rptrGroupIndex c40016
```

- **Port Index**

  An internal or external attribute that returns an integer value when read from the device. It is one of the two index attributes used to access the port table. The attribute ID assigned to Port Index will be used to set the value of `portIndex_Attr` attribute found in the `GnDataRelay_MF` model fragment.

  ```
  CSRF rfc1368Mib rptrPortIndex c4001f
  ```

Port Model Information

The final information needed controls the display of the port models in the Chassis Device and Device Topology Views. The Chassis Device and Device Topology Views display a logical representation of each port. The icon used to display a port in each of these views requires two pieces of information: a counter read from the port and a status read from the port. Attributes from the Port Table, referenced by the Port Index discussed previously, should be used to ensure that the same instance ID is used to read the status and counter for the same port. In the port table, find the following attributes:

- **Counter**

  An internal or external attribute that has the Object Type of counter, integer, or gauge. When read, it will return some accumulating value associated with each port (e.g. number of frames, packets or octets received or transmitted, etc.). The attribute ID assigned to Counter will be used to set the value of `counter_Attr` attribute found in the `GnPortUI_MF` model fragment.

  ```
  CSRF rfc1368Mib rptrMonitorPortReadableFrames c4002e
  ```

- **Status**

  An internal or external attribute that has the Object Type of integer. When read, it will return some operational or administrative status information for the port.The attribute ID

assigned to status will be used to set the value of `status_Attr` attribute found in the `GnPortUI_MF` model fragment.

```
CSRF rfc1368Mib rptrPortOperStatus c4002e
```

- **Status Value**

  Status Value refers to displaying the value read from `status_Attr`. The status value does not have an attribute ID. In the rfc1368 MIB, the `rptrPortOperStatus` object may look something like this:

```
rptrPortOperStatus OBJECT-TYPE

SYNTAXINTEGER {

operational (1),

notOperational (2),

notPresent (3)

}
```

Reading this status will return a value of 1,2,or 3. This information will be used to set the `statusEnum_VTC` attribute in the GnPortUI_MF model fragment. The `statusEnum_VTC` attribute is a text string that provides an enumeration of Value, Text, and Color used in displaying the status information for the port icon in the `statusEnum_VTC` Chassis Device and Device Topology Views.

After you have found the attribute IDs, select **Close Attributes** to exit the **Examine Attributes** view.

Building the Hub Manager Application

Model Type

This section describes creating and configuring a hub manager application model that will be used to model the hub chassis. Additionally, it includes a section on testing the model at this point and a checklist of possible modeling errors.

Creating a Hub Manager Application Model Type

This section describes creating a hub manager application model type using the GnChassisDerPt derivation point. The new hub manager

application model type will be used to model the hub chassis. Follow this procedure:

1. In the Model Type Editor, select **Find Model Type...** from the **File** menu.
2. Enter **GnChassisDerPt** in the **Name or Handle** field.
3. Single-click the **Apply** button to bring up the **GnChassisDerPt** model type.
4. Double-click the **GnChassisDerPt** model type to select it as the current model type.

From this point in the SPECTRUM database, create a new model type that will be used to model the hub manager application.

5. Select **New Model Type** from the **File** menu and name the new model type. For discussion, name the new application **my1368App**. The rfc number and App suffix easily identifies the new model type as an application based on the rfc1368 repeater MIB.
6. Click **OK**.

The new **my1368App** model type should now be the current model type in the Model Type Editor. The `rfc1368Mib` model type should now be added as a base model type to the `my1368App` model type.

7. Select **Add Base Model Type** from the **Edit** menu and add the example MIB model type as a base model type for `my1368App`.
8. Click **Apply** and then **OK**.

The `my1368App` model type should now contain two base model types: GnChassisDerPt and the `rfc1368Mib` MIB model type.

Setting Up the Hub Manager Application Model Type

The new hub manager application model will appear in the Application view as one of many application model types when you model your device with GnSNMPDev. For SpectroSERVER to create the new application model in the Application view, several modifications must be made: setting a text string to be displayed on the Application Icon, setting the `default_attr` attribute, making the new model type instantiable, and defining the hub chassis.

Naming the Hub Manager Application Model

In the Application view, the hub manager application icon will appear with the `my1368App` model type name displayed on zone (c) of the icon (refer to Figure A-1). Zone (a) of the icon can be used to optionally display a user-defined model name. To add a user-defined name to the port model, follow this procedure:

1. In the `my1368App` model type's **Examine Attributes View**, scroll through the **Attribute Names** list until you find the following attribute:

   ```
   CS       Named_EntityModel_Name
   ```

2. Double-click the **Values** field for the `Model_Name` attribute to access the **Alter Value** dialog box.
3. In the **Alter Value** dialog box, enter the name for your port application model (e.g.SNMP Repeater).
4. Click **OK**.
5. Select **Save All Changes** from the **File** menu.

Setting the default_attr Attribute

The new hub manager application model will appear in the Application View as one of many application model types when you model your hub as a GnSNMPDev device. For SpectroSERVER to create the new application model in the Application View, two modifications must be made: setting the `default_attr` attribute and making the new model type instantiable. Follow these steps:

1. In the `my1368App` model type's **Examine Attributes View**, find the `rptrGroupCapacity` attribute and note the attribute ID.

   ```
   DF       rfc1368MibrptrGroupCapacity
   ```

2. Scroll through the `GenCreate_MF` model fragment until you find the `default_attr` attribute.

   ```
   CSCR     Gen_Create_MFdefault_attr
   ```

3. Double-click the **Values** field for `default_attr`.
4. Set the attribute value to the attribute ID from step 1.

5. Make the `my1368App` model type instantiable by clicking the **Instantiable** button and selecting **Save All Changes** from the **File** menu.

The GnSNMPDev model will now discover the new MIB application by determining if an attribute that is characteristic of the new MIB application exists on the device. The GnSNMPDev model looks for the attribute specified in `default_attr`.

Defining the Chassis

Defining the chassis (physical slots and boards) in the `my1368App` model type requires modifying several attributes in the `GnChassis_MF` model fragment, as follows:

1. In the `my1368App` model type's **Examine Attributes View**, scroll down the **Attribute Names** list until you find attributes associated with the `GnChassis_MF` model fragment. For example:

```
SNMP      GnChassis_MFaChassisManager
```

The following attributes in the `GnChassis_MF` model fragment are used to define the physical chassis when Modeling the hub device.

- boardIndex_Attr
- boardType_Attr

2. Find each attribute and double-click the **Values** field to access the **Alter Value** dialog box.
3. In the **Alter Value** dialog box, enter the appropriate Attribute ID that was gathered from the corresponding `rfc1368Mib` model type attributes (refer to the Chassis Information section) and click **OK**. The following table references these attribute.

GnChassis_MF Attribute	rfc1368Mib Attribute	Attribute ID
boardIndex_Attr	rptrGroupIndex	c40016
boardType_Attr	rptrGroupObjectID	c40018

4. Exit the Model Type Editor.

Setting the Data-Relay Functionality

Because the rfc1368 MIB represents a data-relay function, `my1368App` should play a role in determining the device icon symbol. The device icon symbol (also known as a sticky label) is the symbol in the center of the device model that indicates what type of device this model represents. Two model fragments contain the intelligence responsible for determining the device icon symbol of GnSNMPDev. One of these model fragments, `StickyLabelMF`, is a base model type of GnSNMPDev, and the other, `RelayFuncMF`, is a base model type of all application model types that represent some data relay MIB.

Briefly, this is how it works. The `OptionList` attribute of the `StickyLabelMF` determines the precedence and device icon symbol index of each of the possible data relay functions. The list contains relay-function, device icon symbol/index pairs in order from lowest precedence to highest precedence. For GnSNMPDev, this attribute contains:

```
Default,0,SNMP,1,PC,5,WorkStation,7,TServer,6, Repeater,4,
Bridge,2,Router,3,Switch,8
```

This indicates that the terminal server (TServer) functionality is considered more important than WorkStation functionality, but less important than Repeater functionality.

The `RelayFuncMF` model fragment has an attribute, `RelayFunction`, which specifies which data relay function is indicated by this application. The intelligence attached to the `StickyLabelMF` finds the `RelayFunction` of the highest precedence out of all associated applications, and selects the device icon symbol based on that index.

To add this base model type and declare its data relay functionality, do the following:

1. In the Model Type Editor, select **Find Model Type...** from the **File** menu.
2. Enter `my1368App` in the **Name or Handle** field.
3. Single-click the **Apply** button to bring up the `my1368App` model type.
4. Select **Add Base Model Type** from the **Edit** menu and add the `RelayFuncMF` model type as a base model type for `my1368App`.
5. Select **Examine Attributes** from the File menu.
6. Scroll through the `RelayFuncMF` model fragment until you find the `isFunctional` attribute.

```
CSCR     RelayFuncMF‘isFunctional
```

7. Double-click the **Values** field for `isFunctional`
8. Double-click the **Values** field for the `isFunctional` attribute to access the **Alter Value** dialog box.
9. In the **Alter Value** dialog box, enter `TRUE`.
10. Click **OK**.
11. Select **Save All Changes** from the **File** menu.
12. Scroll through the RelayFuncMF model fragment until you find the `RelayFunction` attribute.

```
CSCR     RelayFuncMFRelayFunction
```

13. Double-click the **Values** field for the `RelayFunction` attribute to access the **Alter Value** dialog box.
14. In the **Alter Value** dialog box, enter `Repeater`.
15. Click **OK**.
16. Select **Save All Changes** from the **File** menu.

Testing the Hub Manager Application Model

Before proceeding further, the new hub manager application model should be modeled in SPECTRUM to make sure that the application was built correctly up to this point. The procedure for testing the hub manager application is to model the hub device with the GnSNMPDev model type and then examine the Application view and the Chassis Device view, as follows:

1. Start SpectroSERVER and SpectroGRAPH.
2. Navigate to a **Topology View**.
3. Select **Edit** from the **File** menu.
4. Select the **New Icon...** option from the **Edit** menu to access the **Add Options** dialog box.
5. Select the GnSNMPDev model type from the **Add Options** dialog box and click **OK**. The corresponding **Creation** dialog box is displayed. Fill in the following parameters:

   - Network Address

Enter the Internet Protocol (IP) Address assigned to the hub.

- **Community Name**

  Enter the community name that has been assigned locally to this hub. The default community name value is `public`.

6. After entering the icon parameter information, click **OK**. The icon representing the device model appears in the window and, after a minute or so, the icon will turn green.
7. Return to **View** mode.
8. Highlight the icon and select **Icon Subviews** from the **View** menu.
9. Select **Application** from the **Icon Subviews** menu to access the **Application View**.

If the hub manager application does not appear in the **Application** view, refer to the next section.

If the hub manager application does appear in the **Application** view, exit the **Application** view and access the **Chassis Device** view, as follows:

1. Highlight the icon and select **Icon Subviews** from the **View** menu.
2. Select **Device** from the **Icon Subviews** menu.
3. Select **Chassis** from the **Device** menu.

The **Chassis Device** view provides a logical representation of the hub being modeled. At this point in building the hub manager application, a model of each board residing in the hub should be displayed with the correct slot number. Each board is labeled `GnModule`.

4. Exit the **Chassis Device View**.
5. Select **Edit** from the **File** menu.
6. Highlight the hub model and select **Destroy** from the **Edit** menu to destroy the model.
7. Exit SpectroGRAPH and shut down SpectroSERVER.

If the Test Fails

If the hub manager application you built did not appear as a model in the **Application** view, check the following parameters:

- Make sure that the IP address and community name for the hub being modeled are correct.

- Make sure the `Instantiable` flag is set for the application model type.
- Check the setting of the `default_attr` attribute in the application model type.
- Check the OID Prefix for the MIB model type to ensure that the attribute contains the correct OID address for the device being modeled.

Labeling the Boards

This section describes labeling the boards in the Chassis Device and Device Topology Views. This procedure will replace the `GnModule` label and provide the proper labels on each of the boards displayed in the **Chassis Device** and **Chassis Device Topology** views. Two methods are described: using a descriptor and using a map.

Using a Descriptor

This method requires that an attribute containing the board names exist in the MIB. In our example model, the rfc1368 repeater MIB contains the `rptrGroupDescr` attribute which will supply this information. Labeling the boards in the **Chassis Device** and **Device Topology** views requires setting two attributes in the `GnChassis_MF` model fragment: `boardName_Attr` and `nameParsingCmds`.

> **Note:** Using a descriptor is the preferred method of labeling the boards as it does not require updating the attributes when new types of boards are introduced by the manufacturer.

Follow this procedure:

1. In the Model Type Editor, select **Find Model Type...** from the **File** menu.
2. Enter `my1368App` in the **Name or Handle** field.
3. Single-click the **Apply** button to bring up the `my1368App` model type.
4. Double-click the `my1368App` model type to select it as the current model type.
5. Select **Examine Attributes** from the **File** menu.
6. Scroll down the **Attribute Names** list until you find the `boardName_Attr` attribute.

```
SNMP      GnChassis_MFboardName_Attr
```

7. Double-click the **Values** field for the `boardName_Attr` attribute to access the **Alter Value** dialog box.
8. In the **Alter Value** dialog box, enter the Attribute ID (c40017) for the `rptrGroupDescr` attribute that was found in the **Gathering MIB Information Section** and click **OK**.

Quite often, the text read from the device with this attribute provides more than just the board name. It can also contain a verbose description of the board. For example:

```
Model 3308 10BASE-T Host Module

Model 3301 ThinNet Host Module
```

What should appear in the **Chassis Device** and **Chassis Device Topology** views are the actual board names `3308` and `3301`. To achieve this result, you must set the `nameParsingCmds` attribute. The `nameParsingCmds` attribute supports a sub-set of vi commands that, when set in the `nameParsingCmds` attribute, will edit the descriptor string into a board label.

> **Note:** To use the `nameParsingCmds` attribute, you must first determine the format of the descriptor string on your device. Every vendors' implementation of `rfc1368` will produce strings with their own format. Use an SNMP tool to read this attribute for each board in the hub.

To set the `nameParsingCmds` attribute for the example descriptor, do the following:

1. In the `my1368App` model type's **Examine Attributes View**, scroll down the **Attribute Names** list until you find the `nameparsingCmds` attribute.

   ```
   SNMP     GnChassis_MFnameParsingCmds
   ```

2. Double-click the **Values** field for the `nameParsingCmds` attribute to access the **Alter Value** dialog box.
3. In the **Alter Value** dialog box, enter the following vi commands.

   ```
   dwf D
   ```

This combination of vi commands will delete the first word in the string, leave the board names of `3308` and `3301`, and delete the rest of the string.

4. Select **Save All Changes** from the **File** menu.

At this point, modeling the device with GnSNMPDev would produce actual board names instead of `GnModule`.

The table below provides a list of the vi commands supported by the `nameParsingCmds` attribute.

Editing Commands	**Description**
x	Deletes one character at cursor.
X	Deletes one character to left of cursor.
D	Deletes from cursor to end of line.
d	Deletes one character starting at cursor up to and including the other end specified by cursor motion.
~	Toggles Uppercase/Lowercase. Moves 1 (n) character(s) to the right.
Cursor Motion Commands	**Description**
l	Moves right one character.
h	Moves left one character.
0	Moves left to first character on line.
$	Moves right to last character on line.
fc	Moves right to next character (c).
Fc	Moves left to preceding character (c).
w	Moves right to start of next word.
W	Moves right to start of next “Word”.
e	Moves right to end of next word.
E	Moves right to end of next “Word”.
b	Moves left to preceding start of next word.
B	Moves left to preceding start of next “Word”.

Using a Map

This method can be used when the MIB does not contain a descriptor attribute containing the board names. Creating a board map, requires building an enumeration string in a pre-defined format. Follow this procedure:

1. In the `my1368App` model type's **Examine Attributes** view, scroll down the **Attribute Names** list until you find the `boardName_Map` attribute.

   ```
   SNMP      GnChassis_MFboardName_Map
   ```

2. Double-click the **Values** field for the `boardName_Map` attribute to access the **Alter Value** dialog box.

The value of the `boardName_Map` attribute is read and interpreted by the chassis manager. The text to be entered as the attribute's value is assembled in the following format with each entry separated by a comma.

```
"default name, board type, board name, board
type, board name,......"
```

3. Enter the following information:

   - Default Name

   A text string that is used to label any board for which an entry in the map is not defined. The `boardName_Map` attribute has a Default Name of `GnModule` but other names can be used (e.g. `Unknown`).

   - Board Type

   A representation of the value returned when the board/module type is read from the chassis MIB. The chassis manager will read a value from the chassis MIB and compare it against each Board Type specified in the map. A Board Type entry should be provided for any type of board that can be plugged into the hub being modeled.

   - Board Name

   The text used to label the board model type. The value specified in the map should reflect the manufacturer's product name. A Board Name should be provided for each Board Type specified in the map.

The resulting text string could look something like this example when the Board Type value read from the MIB is an integer:

```
GnModule,4,mm3304-ST,5,mm3305,6,mm3308,......
```

Note: This naming method requires updating the map attribute every time the hub's manufacturer releases a new board type.

Modeling the Repeater Ports

This section describes configuring the hub manager application model to allow modeling the ports on each board. Modeling each port will allow viewing ports in the `Chassis Device` view and allow topology modeling in the `Chassis Device Topology` view. Each port will also be displayed showing a status and a counter.

Defining the Repeater Element

The repeater element of the hub is defined using the `GnDataRelay_MF` model fragment. This model fragment is used to describe the ports found on the boards in the chassis. The `GnDataRelay_MF` model fragment's attributes and intelligence are used by GnSNMPDev to discover, model, and attach port models referenced through the MIB. Follow this procedure:

1. Select the `my1368App` model type as the current model type.
2. Select **Add Base Model Type** from the **Edit** menu and add the `GnRelayDerPt` model type as a base model type for `my1368App`.
3. Click **Apply** and then **OK**.
4. Select **Examine Attributes** from the **File** menu.
5. Scroll down the **Attribute Names** list until you find attributes associated with the `GnDataRelay_MF` model fragment. For example:

```
SNMP     GnDataRelay_MFGnDataRelayMF
```

The following attributes in the `GnDataRelay_MF` model fragment allow SpectroSERVER to find and create all the ports which exist on the boards of the hub being modeled.

- groupIndex_Attr
- portIndex_Attr

6. Find each attribute and double-click its **Values** field to access the **Alter Value** dialog box.
7. In the **Alter Value** dialog box, enter the appropriate Attribute ID that was gathered from the corresponding `rfc1368Mib` model type attributes (refer to the Repeater Information section) and click **OK**. The following table references these attributes.

GnDataRelay_MF Attribute	rfc1368Mib Attribute	Attribute ID
groupIndex_Attr	rptrGroupIndex	c40016
portIndex_Attr	rptrPortIndex	c4001f

Defining the Port Display

The final attribute modifications control the display of the port models in the **Chassis Device** and **Device Topology** views. This requires setting three attributes in the `GnPortUI_MF` model fragment, as follows:

1. In the `my1368App` model type's **Examine Attributes** view, scroll down the **Attribute Names** list until you find attributes associated with the `GnPortUI_MF` model fragment. For example:

```
SNMP      GnPortUI_MFGnPortUIMF
```

The following attributes in the `GnPortUI_MF` model fragment allow SpectroGRAPH to display status information for each port which exist on the boards of the hub being modeled.

- counter_Attr
- status_Attr
- statusEnum_VTC

In the **Alter Value** dialog box, enter the appropriate Attribute ID for the `counter_Attr` and `status_Attr` attributes, that was gathered from the corresponding `rfc1368Mib` model type attributes (refer to the Port Information section) and click **OK**. The following table references these attributes.

GnPortUI_MF Attribute	rfc1368Mib Attribute	Attribute ID
counter_Attr	rptrMonitorPortReadableFrames	c4002e

status_Attr	rptrPortOperStatus	c40022

The `statusEnum_VTC` attribute is a text string that will read the status of the port from the `status_Attr` attribute and display the status in a readable format on the port icon in the **Chassis Device** and **Device Topology** View. The `statusEnum_VTC` attribute maps the value read from the port into a text string to be displayed. Each complete entry in the string has three values: `Value read from device`, `Text to display`, and `Color to use in displaying text`. Every section of the string is separated by a comma (,):

`"value,text,color,value,text,color,value,text,color"`

To construct the `statusEnum_VTC` text string, do the following:

1. Double-click the **Values** field for the `statusEnum_VTC` attribute to access its **Alter Value** dialog box.
2. Add the first `Value read from device` from the `rptrPortOperStatus` MIB object (1) and add a comma (,).

```
rptrPortOperStatus OBJECT-TYPE

   SYNTAX INTEGER {

            operational (1),

            notOperational (2),

            notPresent (3)

          }
```

3. Abbreviate the `Text to display` associated with the previous `Value read from device` to 3 characters or less (e.g. operational could be abbreviated to Opr) and add a comma (,).
4. Add the RGB color that you want associated with the `Text to display` (e.g. 121 = red, 123 = green, 128 = yellow, 154 = blue) as the `Color to use in displaying text` and add a comma (,).
5. Repeat steps 2 through 4 for the remaining values that can be read from the `rptrPortOperStatus` MIB object (2 and 3).

The resulting `statusEnum_VTC` text string could look something like this example:

`1,Opr,123,2,Nop,121,3,NP,154`

6. Select **Save All Changes** from the **File** menu.
7. Exit the Model Type Editor.
8. Start SPECTRUM and create the hub model.

The port icons in the **Chassis Device** and **Device Topology** views will now display the port status as `Opr` (operational) with a green background color when a 1 is read, `Nop` (notOperational) with a red background color when a 2 is read, and `NP` (notPresent) with a blue background color when a 3 is read.

Note: Displaying port icons is not limited to the procedure described in this section. The mechanism demonstrated here is provided to quickly prototype hub ports without having to modify any external SpectroGRAPH files (GIB, PIB, IIB, etc.).

Index

A

Action Mechanism [96]
alarm [107]
Alarm Manager [80]
alert [105]
AlertMap [105]
association [150]

B

boardIndex_Attr [59]

C

Conditional Menu Picks [88]
CsViewControl File [92]

D

default_attr [57]
default_attr_list [57]
Derivable flag [38], [60]
Derivation points [48]
Desc_Key_Word [45]
Device Identification Manager [21]
Device Topology view [99]
Device Type [20]
Device view [94]
Double-Width Boards [103]

E

event [106]
Event Format [107]
EventDisp [106]

G

GIB Editor [93]

GIB files [68]
GnChassis [19]
GnChassis_MF [59]
GnChassisDerPt [51]
GnDevIODerPt [51]
GnModule [61]
GnPort [19], [62]
GnRelayDerPt [51]
GnSNMPAppDerPt [51]

I

IIB files [68]
Image file [68]
image_index [69]
Implementing Conditional Menu Picks [88]
importing a MIB [19]
index file [109]
inference handlers [48]
Instantiable flag [38], [60]
isv file [83]

J

JMib Tools [19]

L

LF_Relations [152]
LF_Rels [152]
Lost and Found [152]

M

Meta-rules [150]
MIB import [19]
mkcd [110]
mkmm [109]
mmbuild [64]
Model Fragment [111]
Model Fragments [59]
model fragments [48]
Model Type Flags [60]

N

No Destroy flag [38], [60]

P

PIB file [67]
Probable Cause [107]

R

read_next [52]
Relations [150]
RelayFuncMF [69]
Required flag [38], [60]

S

Search Manager [80]
SpectroWATCH [20]
sticky label [69]
StickyLabelMF [69], [70]
sysDescr [45]
sysObjectID [45]
SysOIDVerifyList [45]
System_OID_Verify [45]

T

Trap Support [18]

U

Unique flag [38], [60]

V

VCD [47], [63], [109]
Virtual CD [47], [63], [109]
Visible flag [38], [60]